Privacy-preserving genome sequence search.

Kana Shimizu

Department of Computer Science and Engineering
Waseda University

July 23, 2016 @STM 2016

Outline

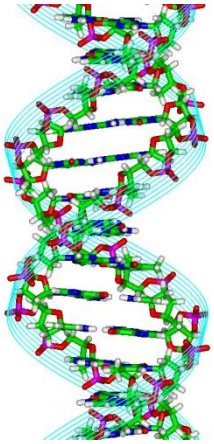
- **Background**
- Additively homomorphic encryption
- Beacon search by Oblivious transfer
- Genome sequence search
  - Overview of the proposed method
  - Recursive oblivious transfer
  - Burrows Wheeler Transform
  - Results
- Conclusion

DNA sequence

- DNA is a molecule that carries genetic information.
- It consists of four nucleotides (Adenine, Guanine, Cytosine, Thymine), thus it is represented as a sequence of four letters.
- Analyzing DNA sequences is one of the most important approaches in current biology.

Next Generation Sequencer

- Recently, the technology for determining DNA sequence has been dramatically improved.
- The instrument that determines DNA sequence based on the new technology is called NGS.



DNA

http://www.pacb.com/

https://www.thermofisher.com/

http://jp.illumina.com/

Sequencer

GGCATGAAA
GTCAGGGCA
GAGCCATCT
ATTGCTTAC
ATTTGCTTC
TGACACAAC
TGTGTTCAC

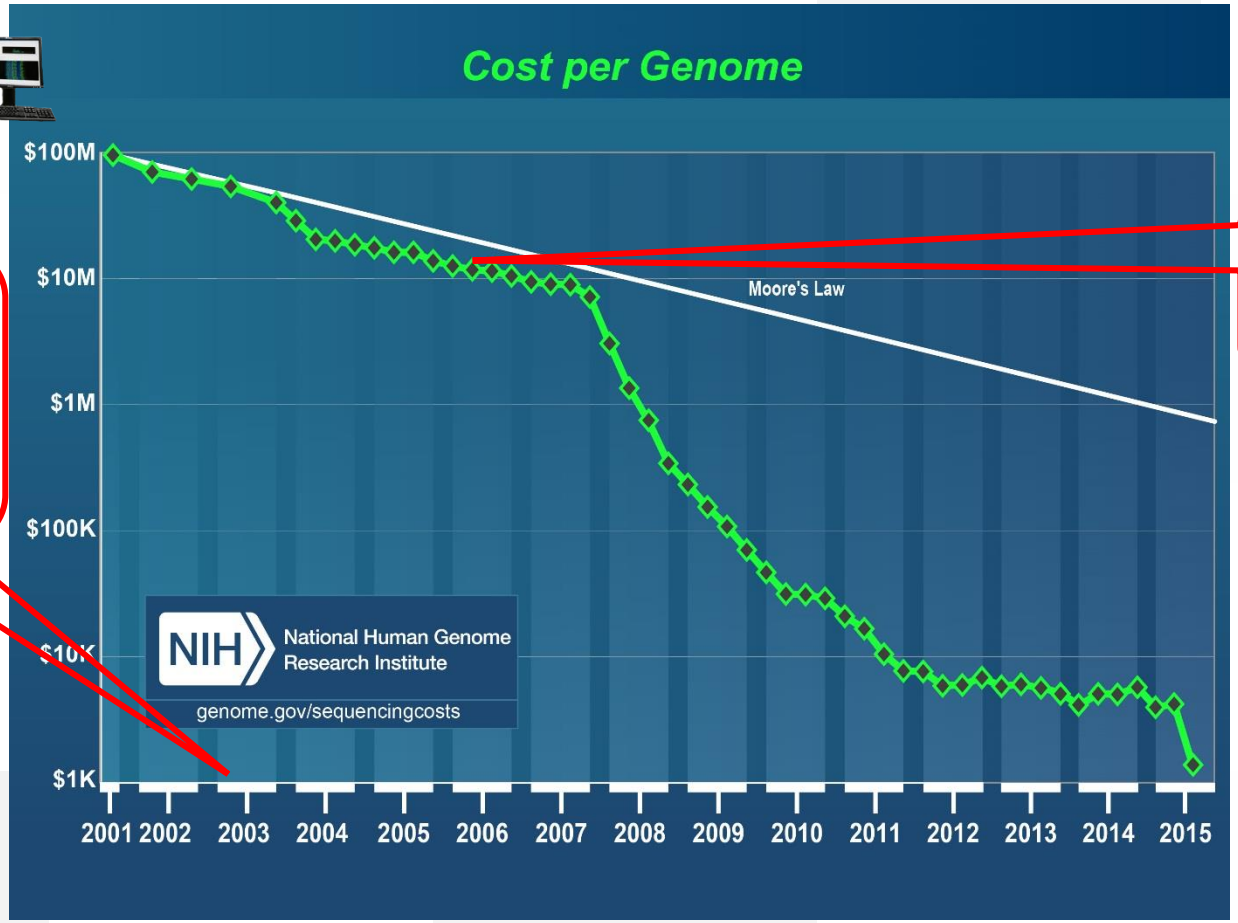
Fragment sequences

Genome “Big Data”

Sanger sequencer

Human Genome Project（1990～2003）

$3B



NGS is introduced to market.

$1K

High-throughput sequencer

http://www.genome.gov/sequencingcosts/

Length of HG: 3・10^9

Growth of personal genome data

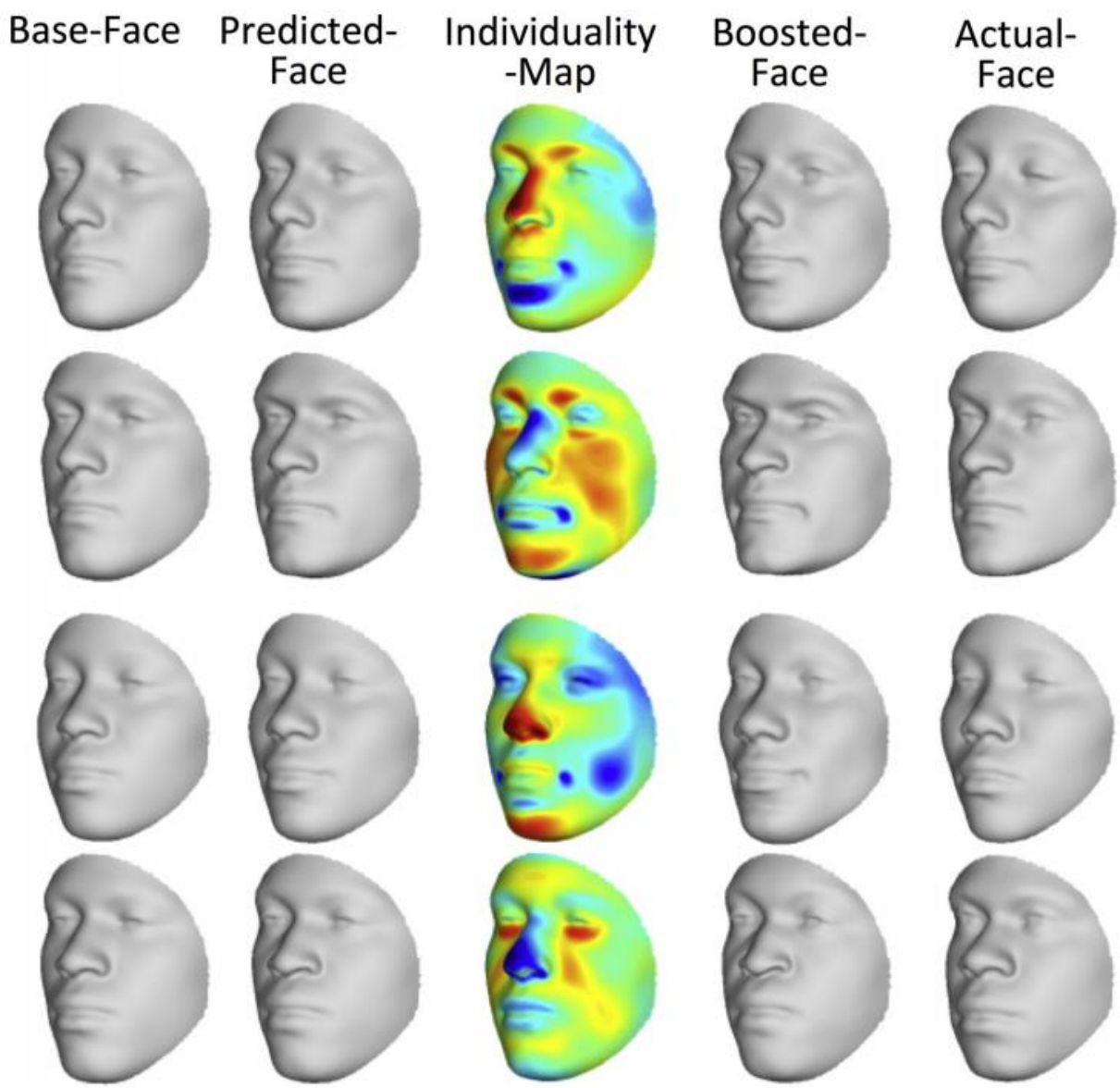
- The huge cost down has encouraged sequencing of individual's genome.
  - Large scale cohort studies such as..
    - ToMMo will recruit 150K participants from 2013 to 2017, in Japan
    - Genomics England aims to sequence 100K individuals' genome, in UK.
  - Direct-to-consumer genetic testing
    - 23andMe has sequenced more than 1M customers' DNA.
    - openSNP: Web site of collecting DTC results
      ≒2700 genotypes (June, 2016)
  - It also poses privacy risks.

Variation of Genome

- The feature of an individual's genome is described by a difference between the genome and the reference genome.

```
Ref:        GGCATGAAAGTCAGGGCAGAGCCATCTATTGC
Individual:GGCATGAAAGTCTGGGCAGAGCCAT TATTGC
```

- Sequence variants are considered to associate with phenotype (observable traits of the individual.)
- Num. of. Known SNP is around 3M
  - SNP: single mutation observed more than 1% of a population.
- One of the important topics of Bioinformatics is to find association between phenotypes and genotypes.
- Some of such associations are already known.
  - BRCA: breast cancer, ADH4: alcohol metabolizing, etc..



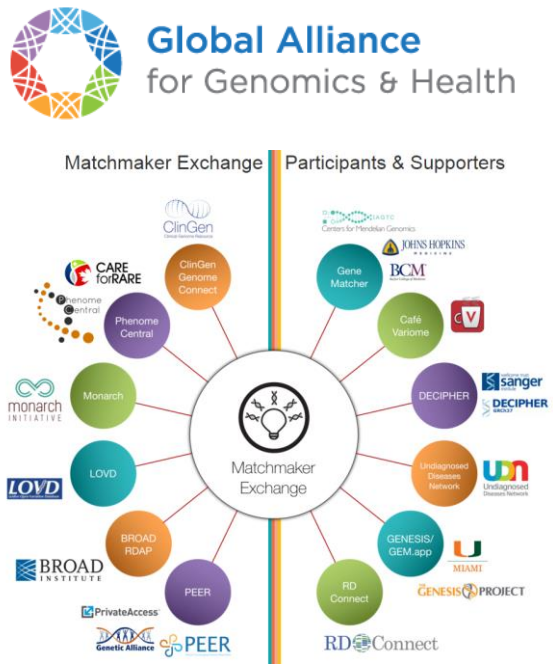
(P. Claes et al. Forensic Science International: Genetics, 2014)

The privacy problems of personal genome

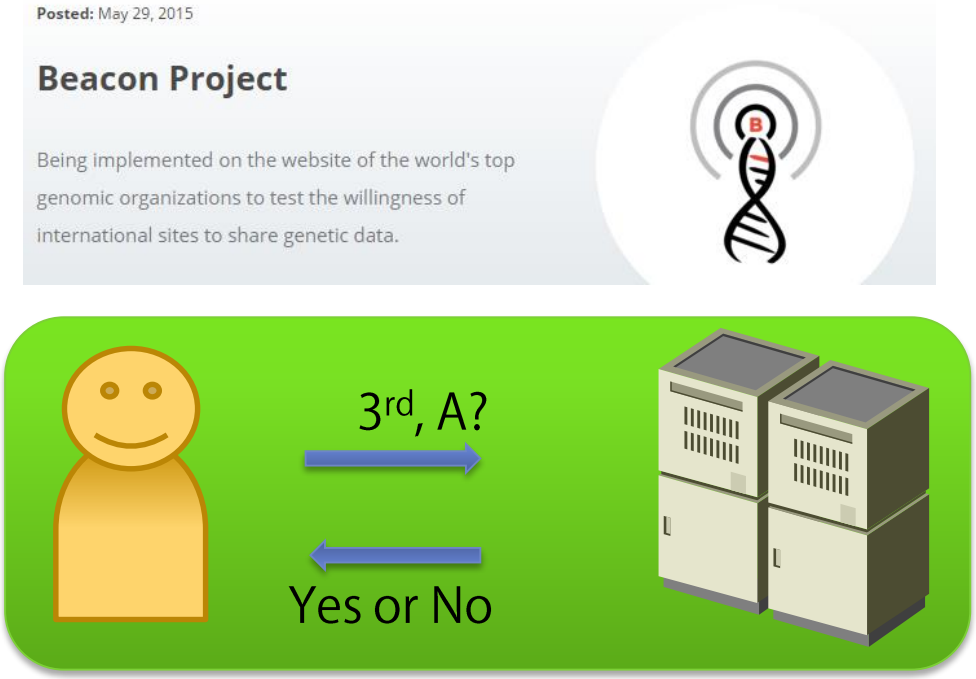
- Genome can be a personal identifier, while it is strongly associated with phenotype.
- Lin et al., 2004
  - ≒80 SNPs can identify an individual.
- Gymrek et al., Science, 2013
  - Surname can be recovered from personal genomes by profiling Y-STRs and querying genetic genealogy databases.
- Homer et al., 2008
  - Statistics of GWAS study leak whether or not a participant belongs to case/control.
- Legislation is not well prepared
  - US: Genetic information nondiscrimination act (GINA)
    - Does not apply to life insurance and the military.
    - The grand daughter of the cancer patient was rejected for the position in US army after taking genetic test (Lindor, 2012)
  - Japan: None
    - Meiji Yasuda Life Insurance Co. is deliberating using people's genetic information to provide targeted services.

The privacy problems of personal genome

- The privacy problem hinders access to many data resources potentially useful for a variety of scientific researches.
- Global Alliance for Genomics & Health
  - Consortium aims for sharing genetic information for research purposes.
  - Established in 2013. 375 institutions has been participated so far.



http://genomicsandhealth.org/



Privacy Preserving Data Mining

- The term PPDM is firstly introduced by the papers (Agrawal & Srikant, 2000) and (Lindell & Pinkas, 2000)
- The goal: To release aggregate information about the data without releasing individual information.
- Example:
  - Aggregate info: Average salary of employees in the University
  - Individual info: A specific employee's salary

Two main approaches

- Perturbation approach
  - The data or the result of the database search is perturbed so that a database user is not able to obtain “true” database contents.
- Cryptographic approach
  - The data holder is called “party”. Each party uses encryption to protect his/her own data. The data are processed without decryption, and only the result of the data mining is obtained by specific parties.
- Those two approaches could be complementary.

Cryptographic approach

- Homomorphic Encryption
  - Enabling add/mul operations on encrypted data.
    - Additive Homomorphic Encryption (Elgamal, 1984), (Paillier, 1999)
    - Fully Homomorphic Encryption (Gentry, 2009)
- Garbled Circuit (Yao, 1986)
  - Enabling computation of any function while the input variables are encrypted.
- Secret Sharing
  - A data point is divided into k shares. The data point is recovered only when θ shares are collected. Some operations can be computed on shares.

Outline

- Background
- **Additively homomorphic encryption**
- Beacon search by Oblivious transfer
- Genome sequence search
  - Overview of the proposed method
  - Recursive oblivious transfer
  - Burrows Wheeler Transform
  - Results
- Conclusion

Homomorphic Encryption

- Homomorphism: A structure-preserving map between two algebraic structures.

$$f:(G,*)\rightarrow(H,\bullet) \quad \text{s.t.} \quad f(g_1 * g_2) = f(g_1)\bullet f(g_2)$$

$$\log:(R_+,\times)\rightarrow(R,+)$$

$$\log(g_1\times g_2) = f(g_1)+f(g_2)$$

- Additive homomorphic encryption
  - Additive op. on the plain text is equivalent to another op. on the cipher text.

$$Enc(m1+m2) = Enc(m1)\oplus Enc(m2)$$

  - Lifted ElGamal [Elgamal84], Paillier [Paillier99]

Additively Homomorphic cryptosystem

- Paillier [Paillier99]

Secret key： $sk = (p, q)$

Public key： $pk = (n, g)\ , n = p \cdot q$

Cipher text of m： $\boxed{Enc_{pk}(m) := g^m \cdot r^n \bmod n^2}$

$r \in Z^*_{n^2}$ is a random value. $g = 1 + kn \bmod n^2$

$$Enc_{pk}(m1) \cdot Enc_{pk}(m2) = g^{m1+m2} \cdot (r1 \cdot r2)^n \bmod n^2$$

$$Dec_{sk}(Enc_{pk}(m1) \cdot Enc_{pk}(m1)) = m1 + m2$$

Secure additive operation based on additive homomorphic encryption

Computing m1 + m2 on the server,
without leaking m1 to the server.

Secret key :
For Decryption only

Public key:
For Encryption only

$m1$

$Enc(m1)$ user's data

$m2$

server's data

Secure additive operation based on additive homomorphic encryption

Secret key :
For Decryption only

Public key:
For Encryption only

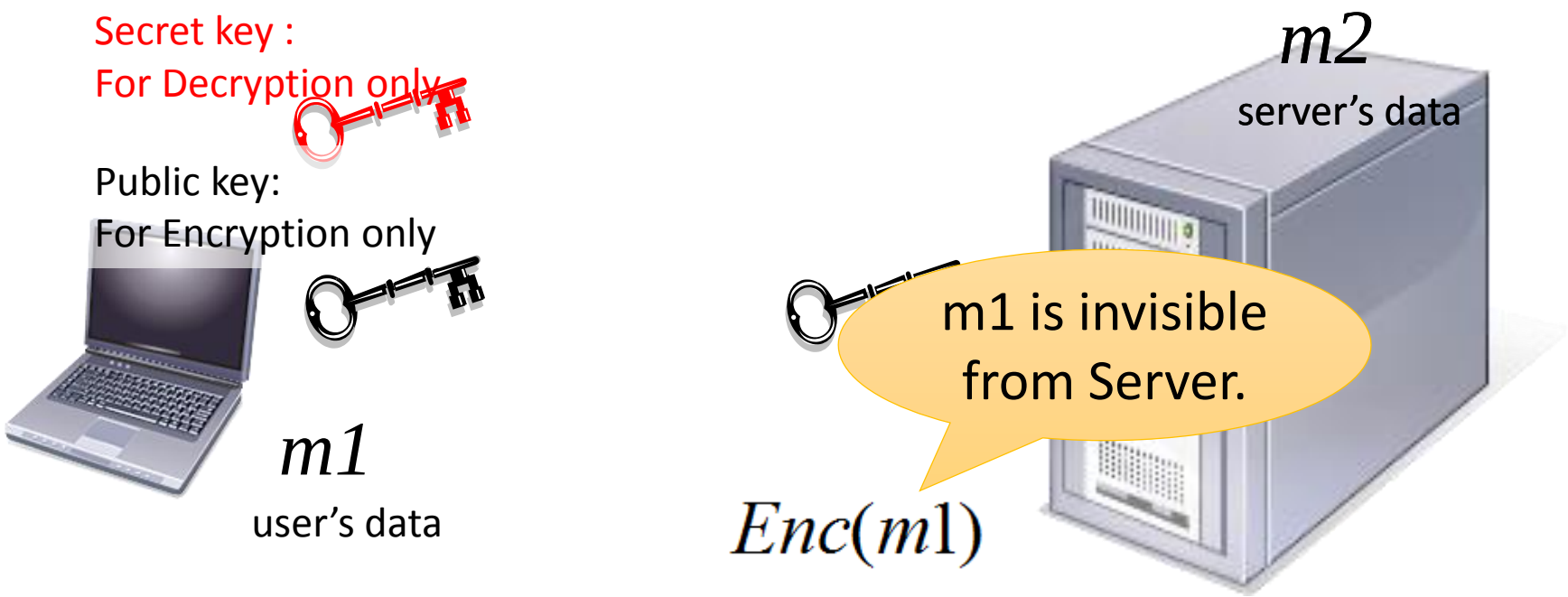
$m1$

$Enc(m1)$ user's data

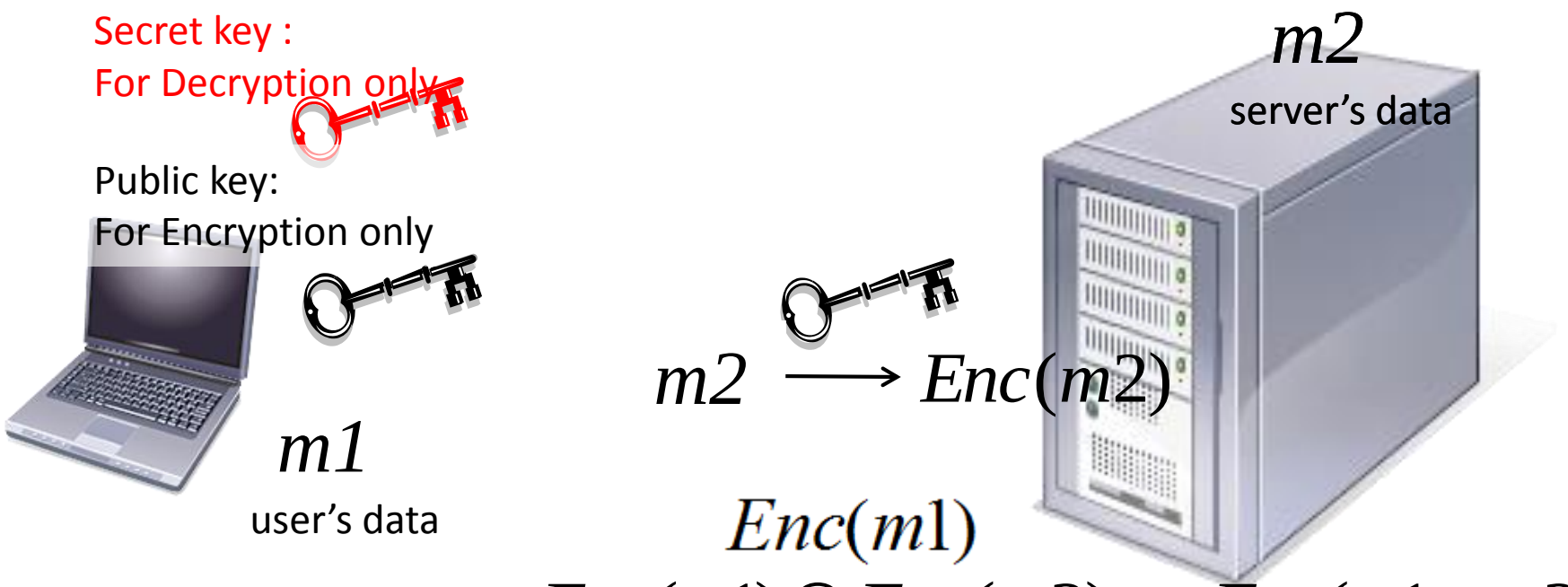
$m2$

server's data

Secure additive operation based on additive homomorphic encryption

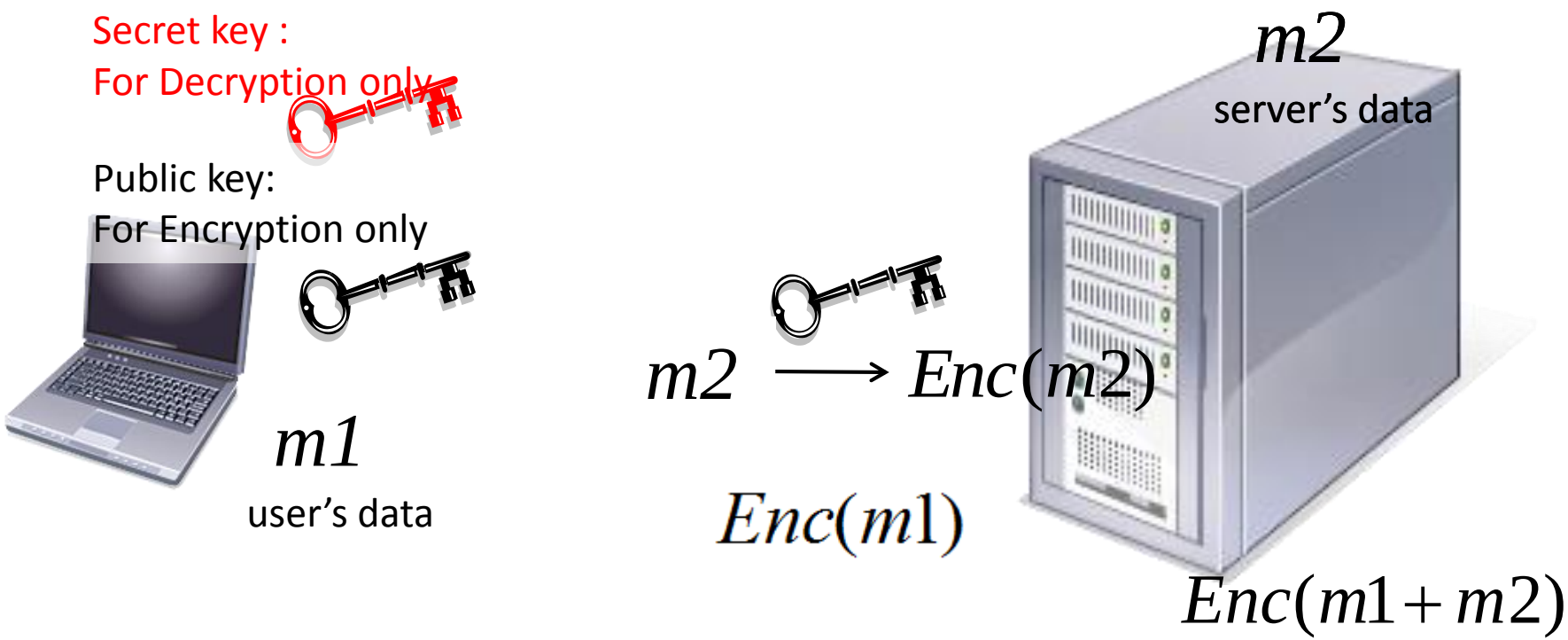


Secure additive operation based on additive homomorphic encryption



$$Enc(m1) \oplus Enc(m2) \rightarrow Enc(m1+m2)$$

Secure additive operation based on additive homomorphic encryption



Secure additive operation based on additive homomorphic encryption

Secret key :
For Decryption only

Public key:
For Encryption only

$m1$

user's data

$Enc(m1+m2)$

$m2$

server's data

Secure additive operation based on additive homomorphic encryption

Secure additive operation based on additive homomorphic encryption

Additive operation is performed on the server without leaking client's value to the server.

Outline

- Background
- Additively homomorphic encryption
- **Beacon search by Oblivious transfer**
- Genome sequence search
  - Overview of the proposed method
  - Recursive oblivious transfer
  - Burrows Wheeler Transform
  - Results
- Conclusion

Can we make secure beacon search?

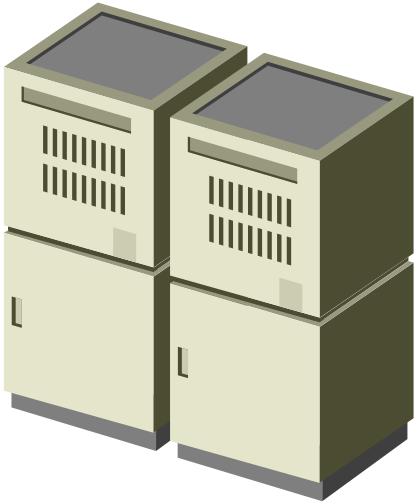
Public | Private

Beacon	Index	Yes: 1 No: 0
1, ‘A’	1	1
1, ‘T’	2	0
1, ‘G’	3	0
1, ‘C’	4	1
2, ‘A’	5	0
…	…	…
3000000000, ‘A’	1199999997	1

Query: (2, ‘A’)

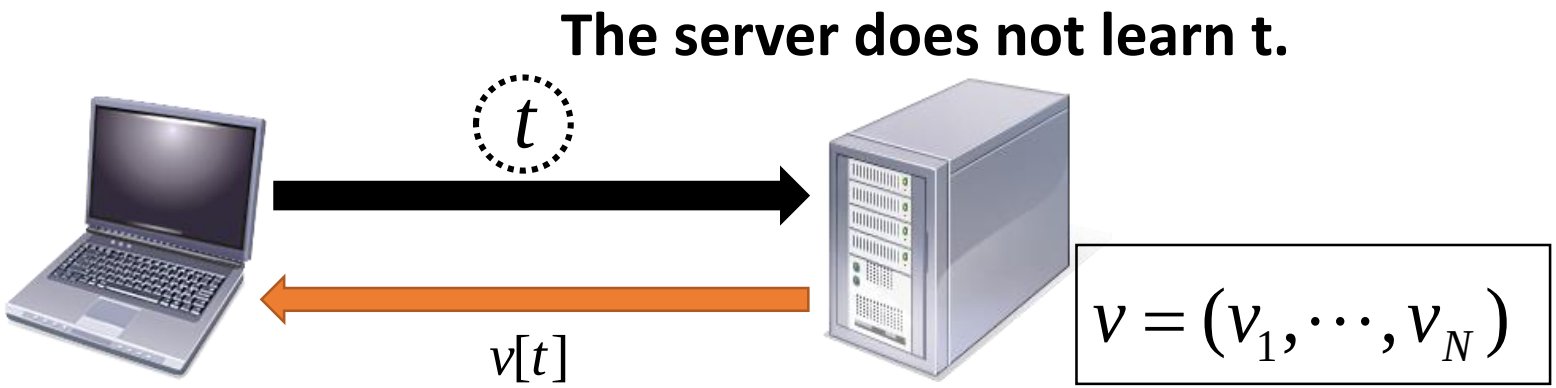
Enc(5)

Enc(0)



What is necessary?

- The user needs to obtain *t*-th element of the server's look-up table (vector) *v* without leaking *t* to the server.
- The problem is conventionally called ***Oblivious Transfer***.



How do we implement OT?

(1 out of N) Oblivious Transfer by AHE

t-th cyphertext

$$\left(\mathrm{Enc}(0),\cdots,\mathrm{Enc}(1),\cdots,\mathrm{Enc}(0)\right)$$

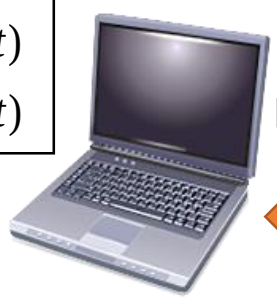
[Step 1] Key setup

Secret key
(for decryption)

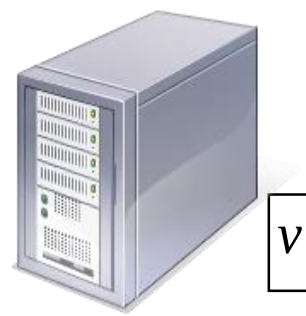
Public key
(for encryption)

[Step 2] Query entry

$$q_i = \begin{cases} 1 & (i = t) \\ 0 & (i \neq t) \end{cases}$$



$$\left(\mathrm{Enc}(q_1),\cdots,\mathrm{Enc}(q_N)\right)$$



$$v = (v_1,\cdots,v_N)$$

(1 out of N) Oblivious Transfer by AHE

Repeat addition of Enc(q[i]) v[i] times

For i = 1,…,N:

$$\mathrm{Enc}(q[i]) \oplus \cdots \oplus \mathrm{Enc}(q[i])$$

[Step 1] Key setup

Secret key (for decryption)

Public key (for encryption)

[Step 3] Computation of an encrypted result

$$\mathrm{Enc}(v[t]) = \left(v[1] \otimes \mathrm{Enc}(q_1)\right) \oplus \cdots \oplus \left(v[N] \otimes \mathrm{Enc}(q_N)\right)$$

$$\because i = t \Rightarrow v[i] \otimes \mathrm{Enc}(q_i) = \mathrm{Enc}(v[i]) \wedge$$

$$i \neq t \Rightarrow v[i] \otimes \mathrm{Enc}(q_i) = \mathrm{Enc}(0)$$

$$q_i = \begin{cases} 1 & (i = t) \\ 0 & (i \neq t) \end{cases}$$

[Step 2] Query entry

$$\left(\mathrm{Enc}(q_1), \cdots, \mathrm{Enc}(q_N)\right)$$

$$v = (v_1, \cdots, v_N)$$

t-th cyphertext

$$\underbrace{\left(v[1] \otimes \mathrm{Enc}(0)\right)}_{\mathbf{Enc(0)}} \oplus \cdots \oplus \underbrace{\left(v[t] \otimes \mathrm{Enc}(1)\right)}_{\mathbf{Enc(v[t])}} \oplus \cdots \oplus \underbrace{\left(v[N] \otimes \mathrm{Enc}(0)\right)}_{\mathbf{Enc(0)}}$$

(1 out of N) Oblivious Transfer by AHE

[Step 1] Key setup

Secret key
(for decryption)

Public key
(for encryption)

[Step 3] Computation of an encrypted result

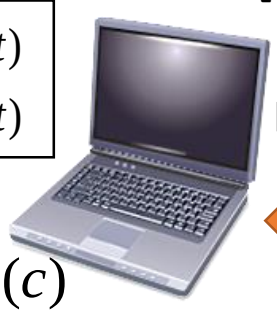
$$\mathrm{Enc}(v[t]) = \big(v[1] \otimes \mathrm{Enc}(q_1)\big) \oplus \cdots \oplus \big(v[N] \otimes \mathrm{Enc}(q_N)\big)$$

$$\because i = t \Rightarrow v[i] \otimes \mathrm{Enc}(q_i) = \mathrm{Enc}(v[i]) \wedge$$

$$i \neq t \Rightarrow v[i] \otimes \mathrm{Enc}(q_i) = \mathrm{Enc}(0)$$

[Step 2] Query entry

$$q_i = \begin{cases} 1 & (i = t) \\ 0 & (i \neq t) \end{cases}$$



$$\big(\mathrm{Enc}(q_1), \cdots, \mathrm{Enc}(q_N)\big)$$

$$v = (v_1, \cdots, v_N)$$

$$c = \mathrm{Enc}(v[t])$$

$$v[t] = \mathrm{Dec}(c)$$

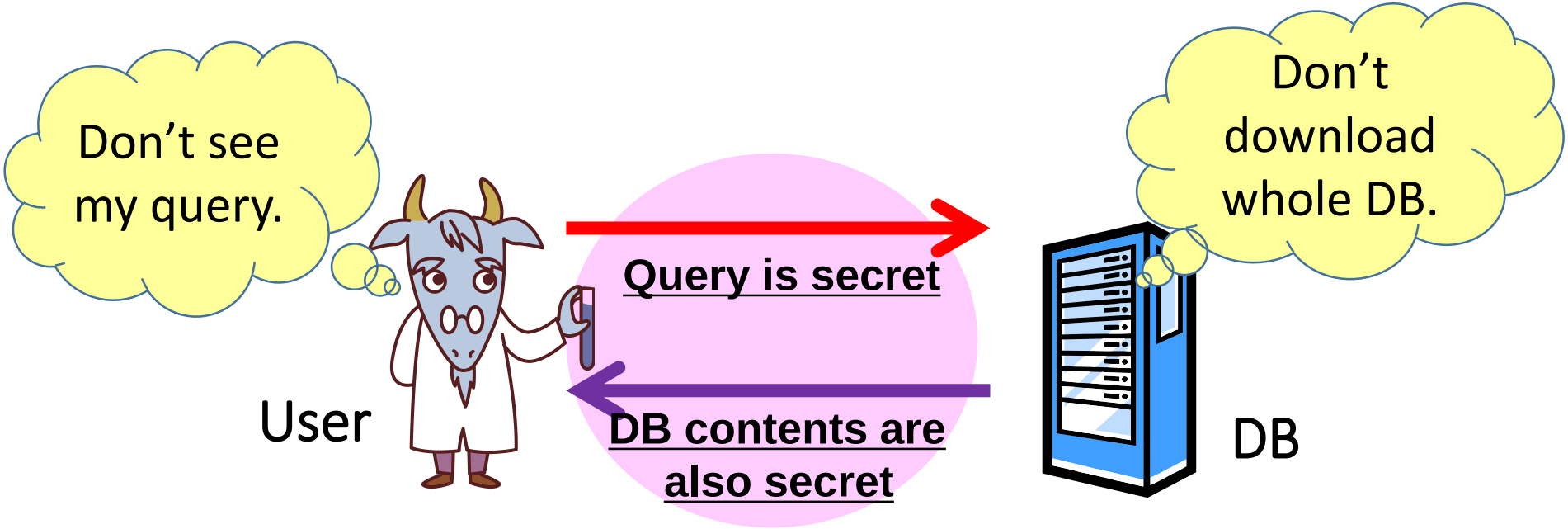
[Step 4] Decryption of the encrypted result

Outline

- Background
- Additively homomorphic encryption
- Beacon search by Oblivious transfer
- **Genome sequence search**
  - Overview of the proposed method
  - Recursive oblivious transfer
  - Burrows Wheeler Transform
  - Results
- Conclusion

Problem Setup

- Our goal is to achieve:
  - A user would like to search a genomic sequence in a database to know whether or not his query matches to the DB.
  - For privacy reasons, the user wants to **conceal the query**, and the server wants to **return only the result**, and do not want to return any other information.



Related Works

- Computation of edit distance (Jha+2008, etc)
- Fixed-length keyword match（Blanton+2010）
- Finding similar sequence based on hamming distance（Baldi+2011, Cristofaro+2013）
- PIR of variable length keyword (Naganuma+2012)

Our goal:

- Search variable length keyword match while keeping both sides' privacy.

Our Approach

- To combine
  - An efficient data structure such as (P)BWT
  - Cryptographic technique (Recursive Oblivious Transfer)

- (P)BWT stores string information very efficiently and still allows computations (Ferragina+2005, Durbin2014)
  - **k-prefix match b/w a query and DB is reported as an interval $[f_k, g_k]$ on the data structure.**
  - **An efficient algorithm is known to compute f_{k+1} from f_k and q[k+1].**
  - Those values are **precomputable**.

(a) Genotype matrix X

	query =	1	0	0			
x0	1	0	0	1	0	1	0	…
x1	1	1	0	1	0	0	1	…
x2	1	0	0	0	0	0	1	…
x3	0	0	0	0	1	1	1	…
x4	1	0	1	1	1	1	0	…

(b) Positional prefix arrays A

x3	x3	x3	x3	x2	x2	…
x0	x0	x0	x2	x0	x1	…
x1	x2	x2	x0	x1	x0	…
x2	x4	x1	x1	x3	x3	…
x4	x1	x4	x4	x4	x4	…

(c) PBWT matrix P

f_1 f_2 f_3

0	0	0	1	0	1	…
0	0	1	0	1	1	…
1	0	0	0	0	0	…
0	1	1	0	1	1	…
0	0	1	1	1	0	…

g_1 g_2 g_3

Searching PBWT by Lookup tables

- The updates can be written in the form of referring a large, static look-up table v.

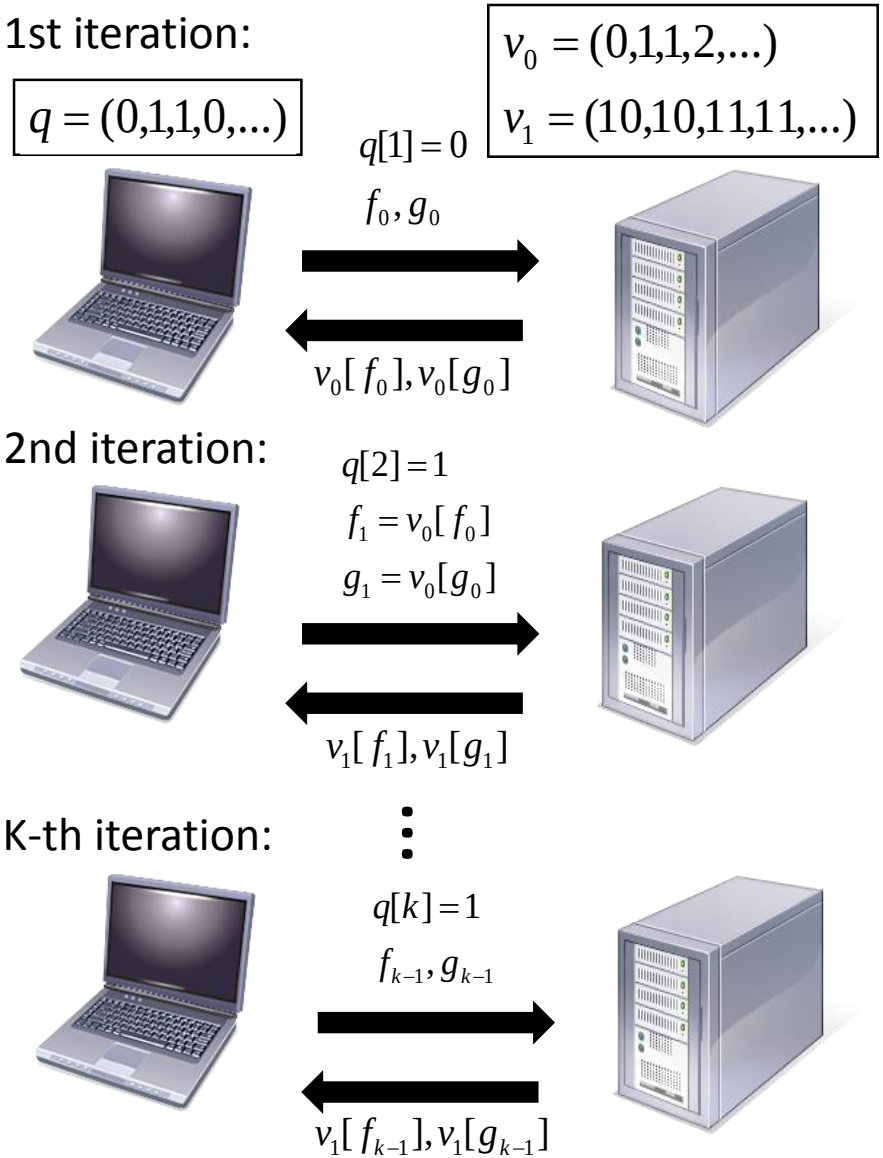
$$f_{K+1} = v_c[f_K]$$

$$g_{K+1} = v_c[g_K]$$

- Match is obtained by:

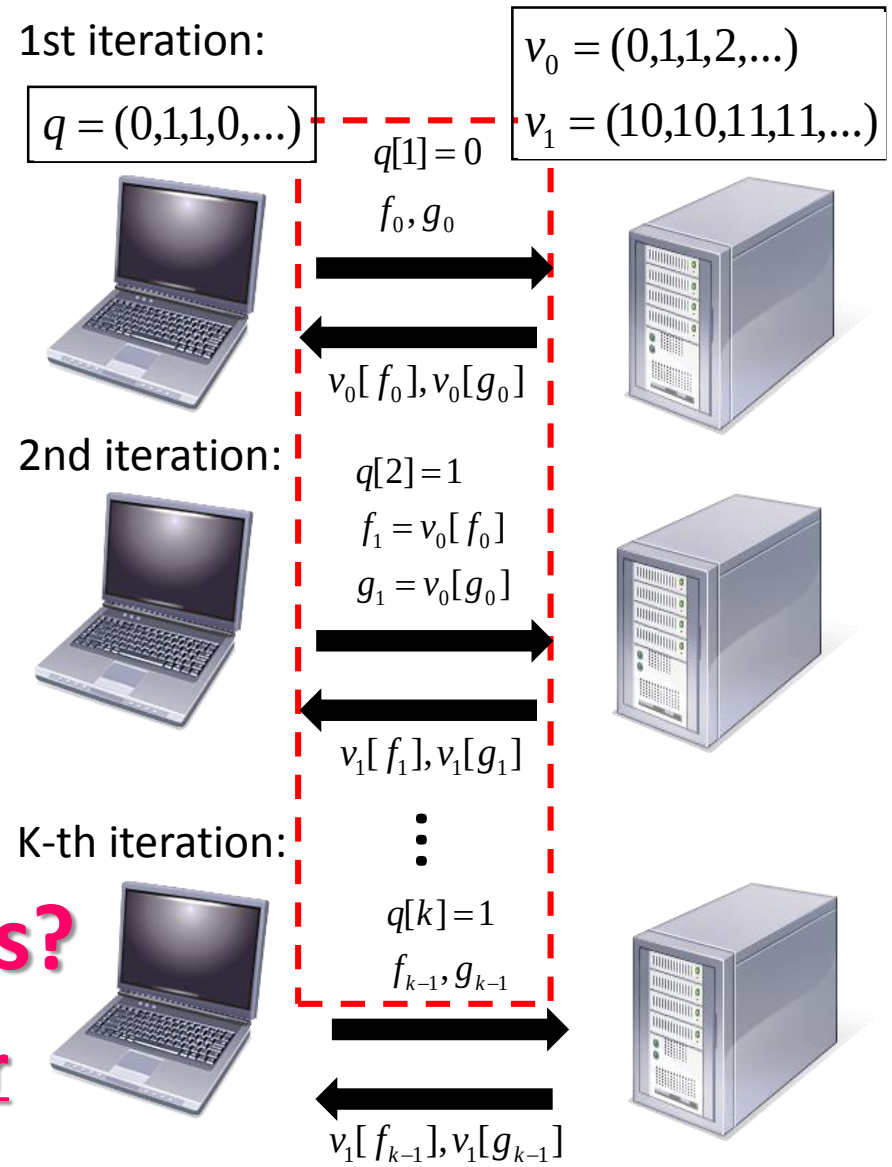
$$g_{K+1} - f_{K+1} + 1$$

OT is used to update f, g



Conceal intermediates

- It is ideal to conceal all the intermediates for protecting server's privacy more rigorously.



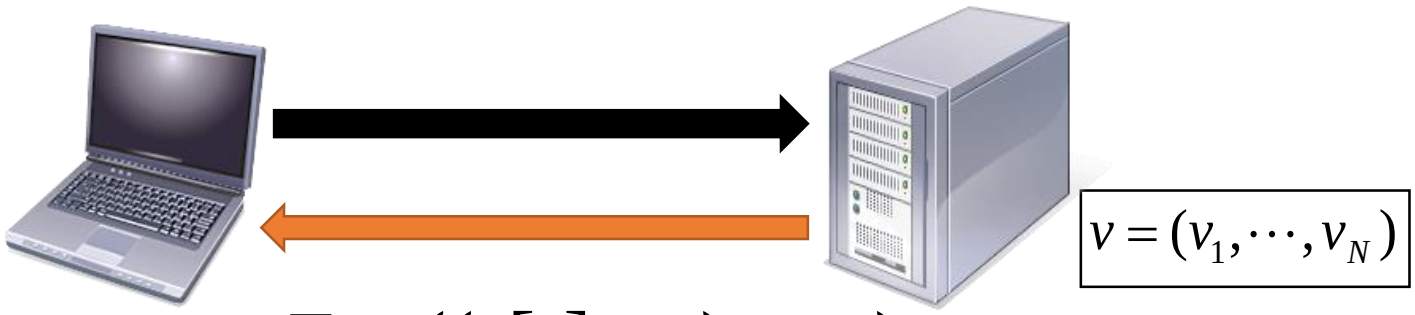
$$f_{k+1} = v_{q[k+1]}[v_{q[k]}[\ldots v_{q[1]}[f_0]\ldots]]$$

$$g_{k+1} = v_{q[k+1]}[v_{q[k]}[\ldots v_{q[1]}[g_0]\ldots]]$$

How do we achieve this?

Recursive Oblivious Transfer

(1 out of N) Recursive OT by AHE



$$c = \mathrm{Enc}((v[t] + \underline{r})_{\mathrm{mod}\, N})$$

Add a random value

(1 out of N) Recursive OT by AHE

(v[t] + r)-th cyphertext

$$\left(\mathrm{Enc}(0),\cdots,\mathrm{Enc}(1),\cdots,\mathrm{Enc}(0)\right)$$

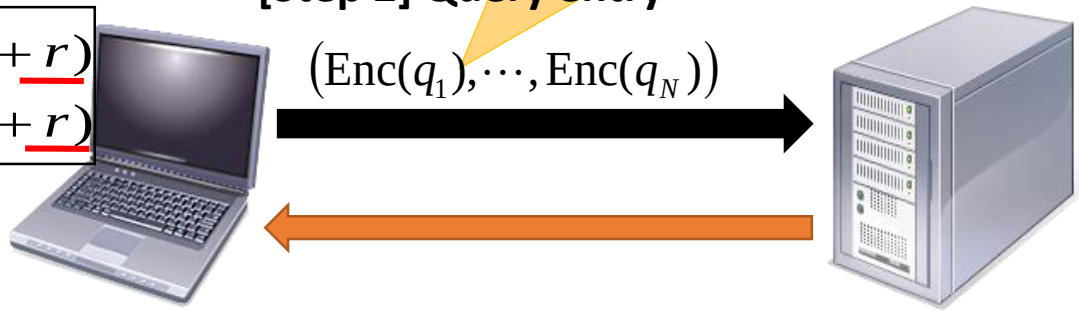
[Step 1] Key setup

Secret key
(for decryption)

Public key
(for encryption)

[Step 2] Query entry

$$q_i = \begin{cases} 1 (i = v[t] + r) \\ 0 (i \neq v[t] + r) \end{cases}$$



$$v = (v_1,\cdots,v_N)$$

(1 out of N) Recursive OT by AHE

Server makes an r-rotated permutation of the query to recover the correct query.

[Step 1] Key setup

Secret key (for decryption)

Public key (for encryption)

[Step 3] Computation of an encrypted result

$$\mathrm{Enc}(v[v[t]]) = \left(v[1] \otimes \mathrm{Enc}(q_{(1-r)\mathrm{Mod\_}N})\right) \oplus \cdots \oplus \left(v[N] \otimes \mathrm{Enc}(q_{(N-r)\mathrm{Mod\_}N})\right)$$

[Step 2] Query entry

$$q_i = \begin{cases} 1 (i = v[t] + r) \\ 0 (i \neq v[t] + r) \end{cases}$$

$$(\mathrm{Enc}(q_1), \cdots, \mathrm{Enc}(q_N))$$

$$v = (v_1, \cdots, v_N)$$

(v[t] + r)-th

$$(\mathrm{Enc}(0), \cdots\cdots\cdots\cdots\cdots\cdots, \mathrm{Enc}(1), \cdots, \mathrm{Enc}(0))$$

v[t]-th

$$(\mathrm{Enc}(0), \cdots, \mathrm{Enc}(1), \cdots\cdots\cdots\cdots\cdots\cdots, \mathrm{Enc}(0))$$

(1 out of N) Recursive OT by AHE

[Step 1] Key setup

Secret key
(for decryption)

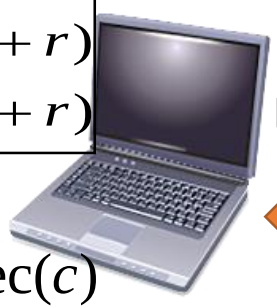
Public key
(for encryption)

[Step 3] Computation of an encrypted result

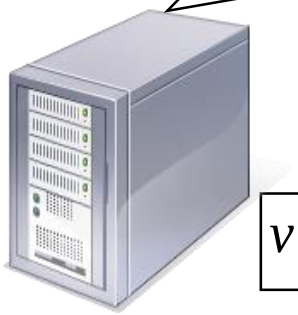
$$\mathrm{Enc}(v[v[t]]) = \left(v[1] \otimes \mathrm{Enc}(q_{(1-r)\mathrm{Mod\_}N})\right) \oplus \cdots \oplus \left(v[N] \otimes \mathrm{Enc}(q_{(N-r)\mathrm{Mod\_}N})\right)$$

[Step 2] Query entry

$$q_i = \begin{cases} 1 (i = v[t] + r) \\ 0 (i \neq v[t] + r) \end{cases}$$



$$(\mathrm{Enc}(q_1), \cdots, \mathrm{Enc}(q_N))$$



$$v = (v_1, \cdots, v_N)$$

$$c = \mathrm{Enc}(v[v[t]])$$

$$v[v[t]] = \mathrm{Dec}(c)$$

[Step 4] Decryption of the encrypted result

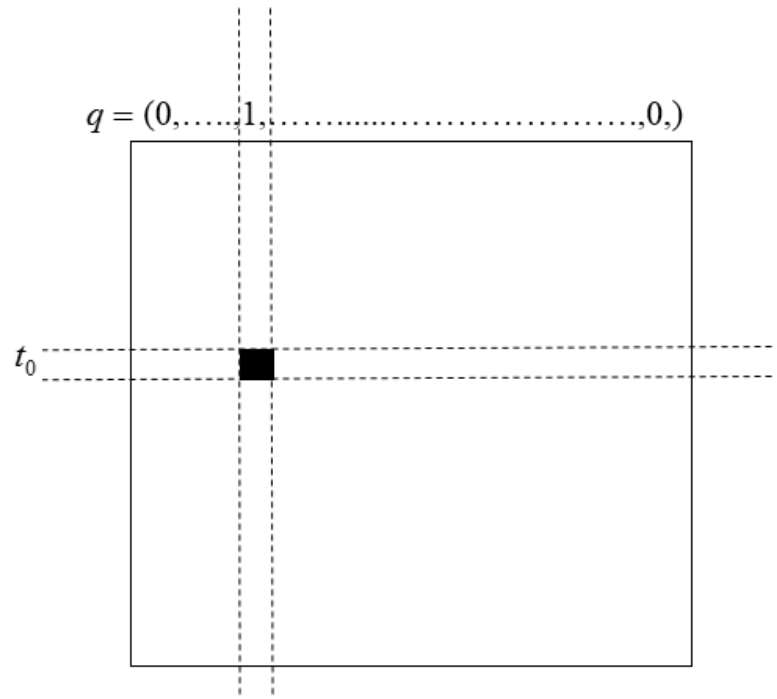
The user obtains v[v[t]] w/o knowing v[t]

A communication efficient algorithm

- Sublinear communication OT (Zhang+2013)
  - O(√N) communication
- Use 2-dimention representation of t :
  - t_0 = t/√N, t_1 = t%√N
  - Computing:
    v[i × √N + t_1] + (t_0 - i) × r

  for i =0 ,…,√N
  - (t_0 - i) × r =0 iff. T_0=i

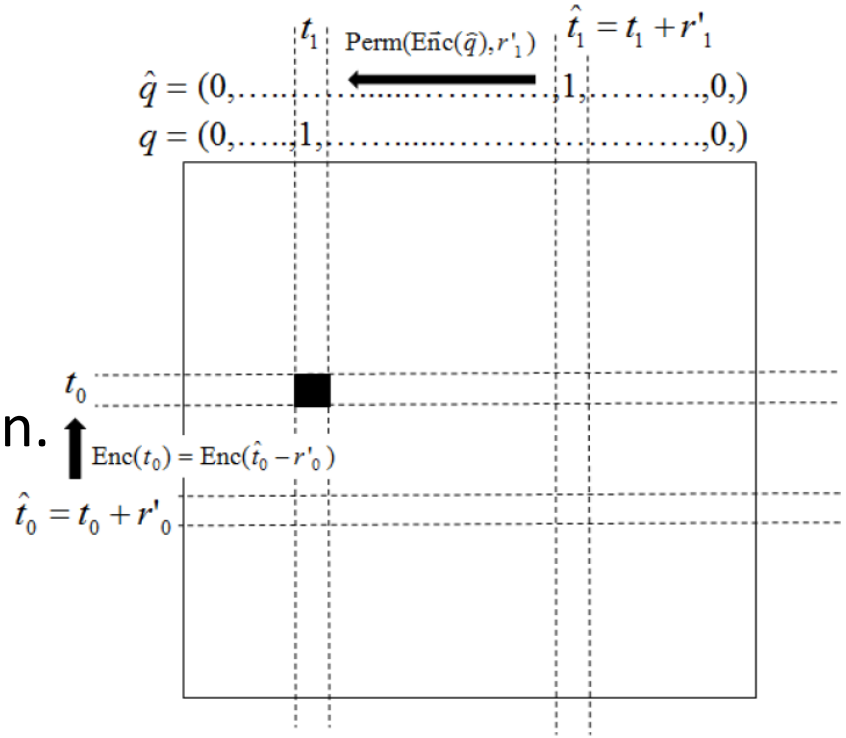
  (can leak v[i × √N + t_1]
  only t_0-th row.)



A communication efficient algorithm

- Sublinear communication OT (Zhang+2013)
  - O(√N) communication
- Use 2-dimention representation of t :
  - t_0 = t/√N, t_1 = t%√N
  - Computing:
    v[i × √N + t_1] + (t_0 - i) × r

  for i =0 ,…,√N
  - Use similar technique
    to design a recursive version.



Recursive search data structure for genomic data

- Our approach is applicable for the data structure enabling recursive search such as..
- BWT (Burrows+94, Ferragina+00)
  - A popular algorithm for NGS read alignment.
  - BWA(Li&Durbin10)
  - Bowtie(Langmead+09)
  - SOAP(Li+08)

  etc…
- PBWT (Durbin14)
  - BWT like structure for searching aligned sequences.

Preparation

- i-th character of a string S is denoted by S[i].
- Rank dictionary:

$$\mathrm{Rank}_C(S,t) = \left|\{ j \mid S[j] = c, \quad 1 \le j \le t \}\right|$$

(Example)

```
i:123456789
S:ATGCTAGCT
```

$$\mathrm{Rank}_{\mathrm{A}}(S,6) = 2$$

$$\mathrm{Rank}_{\mathrm{T}}(S,3) = 1$$

$$\mathrm{CF}_c(S) = \sum_{r<c} \mathrm{Rank}_r(S,N)$$

$$\mathrm{CF}_{\mathrm{A}}(S) = 0$$

$$\mathrm{CF}_{\mathrm{T}}(S) = 6$$

Suffix Array (Manber91)

- Sorted array of all suffixes of a string.

```
S="ATGAATGCGA$"
```

					SA
S1	ATGAATGCGA$		S11	$	11
S2	TGAATGCGA$		S10	A$	10
S3	GAATGCGA$		S4	AATGCGA$	4
S4	AATGCGA$		S1	ATGAATGCGA$	1
S5	ATGCGA$		S5	ATGCGA$	5
S6	TGCGA$		S8	CGA$	8
S7	GCGA$		S9	GA$	9
S8	CGA$		S3	GAATGCGA$	3
S9	GA$		S7	GCGA$	7
S10	A$		S2	TGAATGCGA$	2
S11	$		S6	TGCGA$	6

Searching on SA

- **Conduct binary search.**

(Example) Search “ATG”.

```
                    S11 $
                    S10 A$
                    S4  AATGCGA$
                    S1  ATGAATGCGA$
                    S5  ATGCGA$
Greater than “ATG”? S8  CGA$
                    S9  GA$
                    S3  GAATGCGA$
                    S7  GCGA$
                    S2  TGAATGCGA$
                    S6  TGCGA$
```

Searching on SA

- **Conduct binary search.**

(Example) Search “ATG”.

Greater than “ATG”?

S11	$
S10	A$
S4	AATGCGA$
S1	ATGAATGCGA$
S5	ATGCGA$
~~S8~~	~~CGA$~~
~~S9~~	~~GA$~~
~~S3~~	~~GAATGCGA$~~
~~S7~~	~~GCGA$~~
~~S2~~	~~TGAATGCGA$~~
~~S6~~	~~TGCGA$~~

Searching on SA

- **Conduct binary search.**

(Example) Search “ATG”.

Greater than “ATG”?

~~S11 $~~
~~S10 A$~~
~~S4 AATGCGA$~~
S1 ATGAATGCGA$
S5 ATGCGA$
~~S8 CGA$~~
~~S9 GA$~~
~~S3 GAATGCGA$~~
~~S7 GCGA$~~
~~S2 TGAATGCGA$~~
~~S6 TGCGA$~~

Searching on SA

- **Conduct binary search.**

(Example) Search “ATG”.

Greater than “ATG”?

~~S11 $~~
~~S10 A$~~
~~S4 AATGCGA$~~
S1 ATGAATGCGA$
S5 ATGCGA$
~~S8 CGA$~~
~~S9 GA$~~
~~S3 GAATGCGA$~~
~~S7 GCGA$~~
~~S2 TGAATGCGA$~~
~~S6 TGCGA$~~

Searching on SA

- **Conduct binary search.**

(Example) Search “ATG”.

Greater than “ATG”?

~~S11 $~~
~~S10 A$~~
~~S4 AATGCGA$~~
S1 ATGAATGCGA$
S5 ATGCGA$
~~S8 CGA$~~
~~S9 GA$~~
~~S3 GAATGCGA$~~
~~S7 CGGA$~~

An efficient construction of SA (Nong+09):
O(N) time, O(N(log N+log|Σ|))space

Burrows-Wheeler Transform (Burrows+94)

- B[i] = S[SA[i]-1]

```
S="ATGAATGCGA$"

A   $
G   A$
G   AATGCGA$
$   ATGAATGCGA$
A   ATGCGA$
G   CGA$
C   GA$
T   GAATGCGA$
T   GCGA$
A   TGAATGCGA$
A   TGCGA$
```

Burrows-Wheeler Transform (Burrows+94)

- B is *Reversible* transformation of S
  - No need to store additional data.
- Good fit to compression
  - Identical characters tends to be near.
- Searchable (FM-index)

Let’s start from an extreme case

B[i] = S[SA[i]-1]

S="bfcgahejid$"

S[SA[1]=11]

B[1] = S[11-1]

d	←	$
g	←	ahejid$
$	←	bfcgahejid$
f	←	cgahejid$
i	←	d$
h	←	ejid$
b	←	fcgahejid$
c	←	gahejid$
a	←	hejid$
j	←	id$
e	←	jid$

Let’s start from an extreme case

B[i] = S[SA[i]-1]

S="bfcgahej**id$**"

d ←	**$**
g ←	**a**hejid$
$ ←	**b**fcgahejid$
f ←	**c**gahejid$
i ←	**d**$
h ←	**e**jid$
b ←	**f**cgahejid$
c ←	**g**ahejid$
a ←	**h**ejid$
j ←	**i**d$
e ←	**j**id$

Let’s start from an extreme case

B[i] = S[SA[i]-1]

S="bfcgahe**jid$**"

d ←	**$**
g ←	**a**hejid$
$ ←	**b**fcgahejid$
f ←	**c**gahejid$
i ←	**d**$
h ←	**e**jid$
b ←	**f**cgahejid$
c ←	**g**ahejid$
a ←	**h**ejid$
j ←	**i**d$
e ←	**j**id$

Let’s start from an extreme case

B[i] = S[SA[i]-1]

S="bfcgah**ejid$**"

d	←	**$**
g	←	**a**hejid$
$	←	**b**fcgahejid$
f	←	**c**gahejid$
i	←	**d**$
h	←	**e**jid$
b	←	**f**cgahejid$
c	←	**g**ahejid$
a	←	**h**ejid$
j	←	**i**d$
e	←	**j**id$

What about identical characters?

S="ATGAATGCGA$"

A	**$**
G	**A**$
G	**A**ATGCGA$
$	**A**TGAATGCGA$
A	**A**TGCGA$
G	**C**GA$
C	**G**A$
T	**G**AATGCGA$
T	**G**CGA$
A	**T**GAATGCGA$
A	**T**GCGA$

What about identical characters?

S="ATGAATGCGA$"

A_1	$
G	$A_1$$
G	A_2ATGCGA$
$	A_3TGAATGCGA$
A_2	A_4TGCGA$
G	C GA$
C	G A$
T	G AATGCGA$
T	G CGA$
A_3	T GAATGCGA$
A_4	T GCGA$

The position of the same 'A' is determined by the same substring.

What about identical characters?

S="ATGAATGC**GA$**"

A_1	$
G	$A_1$$
G	A_2ATGCGA$
$	A_3TGAATGCGA$
A_2	A_4TGCGA$
G	C GA$
C	G A$
T	G AATGCGA$
T	G CGA$
A_3	T GAATGCGA$
A_4	T GCGA$

The position of the same 'A' is determined by the same substring.

What about identical characters?

S="ATGAATG**CGA$**"

LF-Mapping:

$$\mathrm{LF}(i) := \mathrm{CF}_{\mathrm{B}[i]}(\mathrm{B}) + \mathrm{Rank}_{\mathrm{B}[i]}(\mathrm{B}, i)$$

```
P = 1
for i = 1 to N
  S[N-i] = B[p]
  p = LF(p)
end for
```

A_1	$
G_1	$A_1$$
G_2	A_2ATGCGA$
$	A_3TGAATGCGA$
A_2	A_4TGCGA$
G_3	C GA$
C	G_1A$
T	G_2AATGCGA$
T	G_3CGA$
A_3	T GAATGCGA$
A_4	T GCGA$

The position of the same 'A' is determined by the same substring.

FM-index (Ferragina+00)

- Searching on BWT
  - Using a rank dictionary on BWT of S

- Backward search
  - Searching from the last character and extend the match one by one, in similar way to LF-mapping.
  - Time complexity
    - O(1) by a space consuming dictionary, O(Log |Σ|) by Wavelet tree (Grossi+03).

$$f' = \mathrm{CF}_C(\mathrm{B}) + \mathrm{Rank}_C(\mathrm{B}, f-1) + 1$$

$$g' = \mathrm{CF}_C(\mathrm{B}) + \mathrm{Rank}_C(\mathrm{B}, g) + 1$$

FM-index (Ferragina+00)

S="ATGAATGCGA$"

Extending the match by 'G' from "A$".

```
i = |q|
f=1, g=N
While f<=g
   c = q[i--]
```

$$f = \mathrm{CF}_C(\mathrm{B}) + \mathrm{Rank}_C(\mathrm{B}, f-1) + 1$$

$$g = \mathrm{CF}_C(\mathrm{B}) + \mathrm{Rank}_C(\mathrm{B}, g) + 1$$

```
end for
```

$f = 2,\ g = 5 \ \rightarrow\ f' = 7,\ g' = 8$

A_1	$
G_1	**A_1**$
G_2	**A_2**ATGCGA$
$	**A_3**TGAATGCGA$
A_2	**A_4**TGCGA$
G_3	C_1GA$
C	**G_1**A$
T	**G_2**AATGCGA$
T	G_3CGA$
A_3	T_1GAATGCGA$
A_4	T_1GCGA$

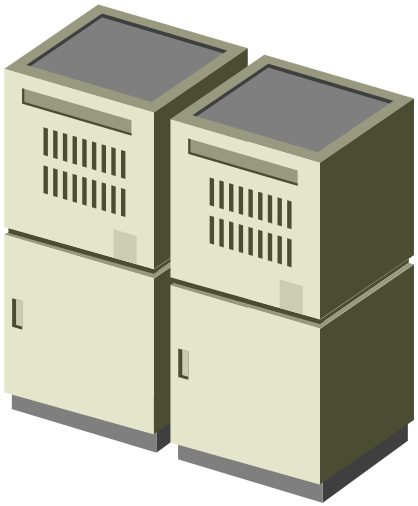
Searching on aligned sequences

Match is computed by the data structure (pBWT) similar to BWT.

```
query:  GCA…GAAA
  s1:  ATGCA…AGCTA
  s2:  ATGTC…TATGT
  s3:  TTGCC…AGCGA
  s4:  TTGTC…TATGT
  s5:  GTGCA…GACTA
  s6:  CTGTC…TATGT
  …
  sM:  CTGTC…TATGT
```

Sequence
k

# of matches from k-th SNP for every prefix



PBWT-sec (shimizu+2016)

- PBWT (Durbin, 2014) + Recursive OT

Algorithm: PBWT-sec

Server creates a look-up table V
User initialize [f, g]
for k = 1 ,…, L:
    // updating [f, g]
    User sends $f = f + q[k] \times M$, $g = g + q[k] \times M$
    Server returns V(f, k), V (g, k)
    User updates f = V(f, k), g = V(g, k)
    User knows # of k-prefix matches by (g – f + 1)
    if g – f < 0: then exit;

Computing matches by ROT

Shimizu+, Bioinformatics, 2016
https://github.com/iskana/PBWT-sec

Complexity

- PBWT-sec
  - Linear to the query length l
- Standard (exhaustive) approach
  - Sending every suffix of a query to check matches
  - Exponential to the query length l

	Time	Communication	Space
CP (user)	$O(\ell\sqrt{MD\lvert\Sigma\rvert})$	$O(\ell\sqrt{MD\lvert\Sigma\rvert})$	$O(\sqrt{MD\lvert\Sigma\rvert})$
CP (server)	$O(\ell MD\lvert\Sigma\rvert)$	$O(\ell\sqrt{MD\lvert\Sigma\rvert})$	$O(MD\lvert\Sigma\rvert)$
EX (user)	$O(\sqrt{D\lvert\Sigma\rvert^{\ell}})$	$O(\sqrt{D\lvert\Sigma\rvert^{\ell}})$	$O(\sqrt{D\lvert\Sigma\rvert^{\ell}})$
EX (server)	$O(D\lvert\Sigma\rvert^{\ell})$	$O(\sqrt{D\lvert\Sigma\rvert^{\ell}})$	$O(D\lvert\Sigma\rvert^{\ell})$

※ Alphabet friendly algorithm has been developed (Sudo+, *in preparation*)

Experimental setup

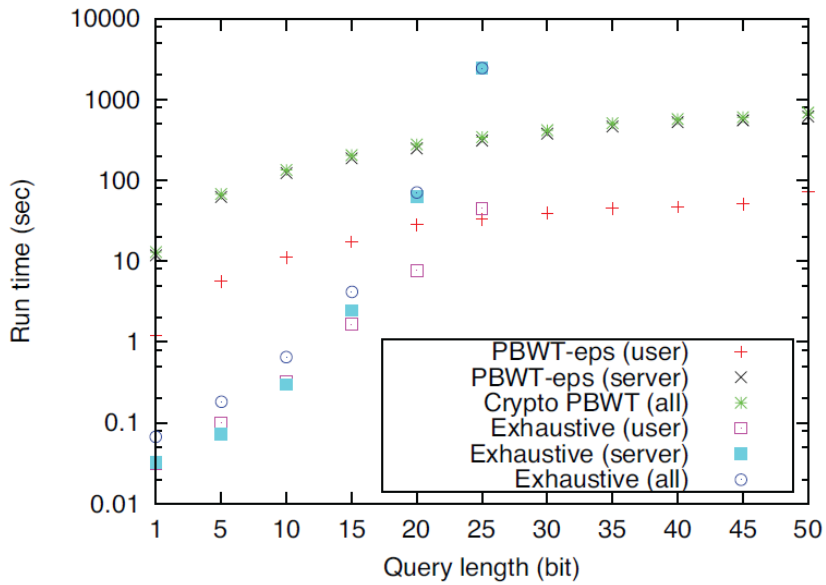
- Implementation of PBWT-sec
  - C++ using AISTCRYPT (Open source C++ library of EC Elgamal).

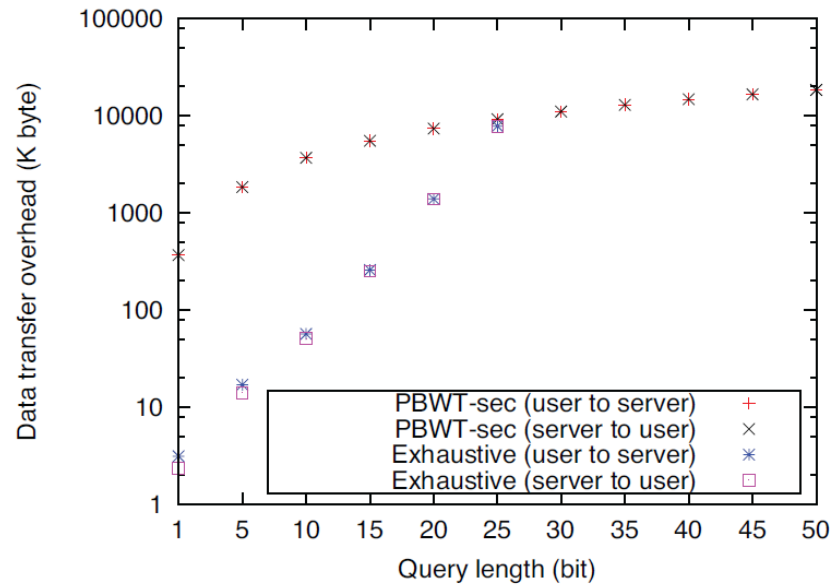
- 2,184 haploid genomes from the chrom. 1 of the 1,000 Genomes Project (phase 1 data release).

- Tested on:
  - Laptop
    (Intel Core(TM) i7 3.00GHz CPU; total 4 cores with HT)
  - A compute node
    (Intel Xeon 2.40GHz CPU; total of 32 cores with HT)

Performance on laptop computers

- The observed run time and data transfer size of PBWT-sec is linear in the query length, while that of the exhaustive approach is exponential.





Run time

- Combined user's and server's run time was 15 sec for searching on 2,184 genomes by laptop (D=1)
- A compute node took between 7 and 132 seconds depending on the level of privacy.

	Laptop	Compute node		
Parallel Compute Cores	4	4	8	16
Run time (sec) with D = 1	15	22	15	7
Run time (sec) with D = 5	43	47	39	18
Run time (sec) with D = 10	78	84	68	31
Run time (sec) with D = 20	141	154	113	56
Run time (sec) with D = 50	338	386	260	132

D is a parameter for privacy level of the server.

Conclusion

- We have proposed a novel approach for searching genomic sequences in a privacy-preserving manner.
- It achieves high utility and has strong security features and requires acceptable compute and communication resources.
- The algorithm can be used to facilitate sharing of genetic information across institutions and countries in order to identify large enough cohorts with a similar genetic backgrounds.

Acknowledgements

- Co-auhtors
  - Gunnar Rätsch (ETHZ)
  - Koji Nuida (AIST)

- Shigeo Mitsunari (Cybozu) for developing AISTCRYPT.

- Members of Computational Biology Research Center at AIST