ATTRIBUTE COMPARSION IN CP-ABE

B.TECH PROJECT PRESENTATION

SUPERVISOR : DR. PADMALOCHAN BERA

Prateek Kumar Singh
20CS01003

WHAT IS ABE?

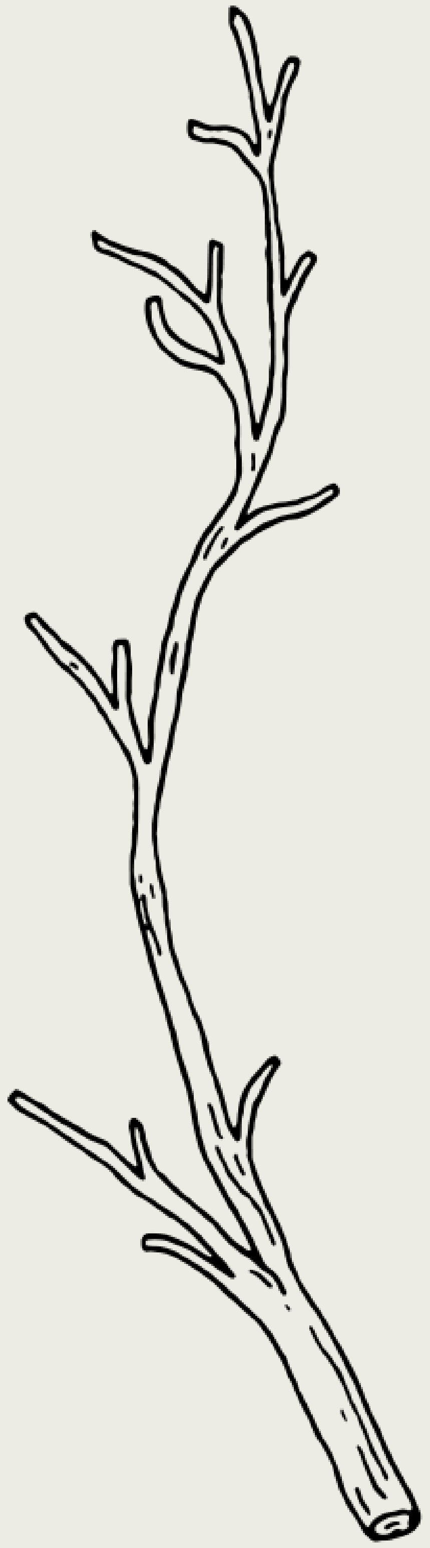
Attribute-based encryption

- It is a generalisation of public-key encryption which enables fine grained access control of encrypted data using authorisation policies.

- The secret key of a user and the ciphertext are dependent upon attributes.

- Decryption of a ciphertext is possible only if the set of attributes of the user key matches the attributes of the ciphertext

TYPES OF ABE



KEY-POLICY ATTRIBUTE-BASED ENCRYPTION

In KP-ABE, users' secret keys are generated based on an access tree that defines the privileges scope of the concerned user, and data are encrypted over a set of attributes.

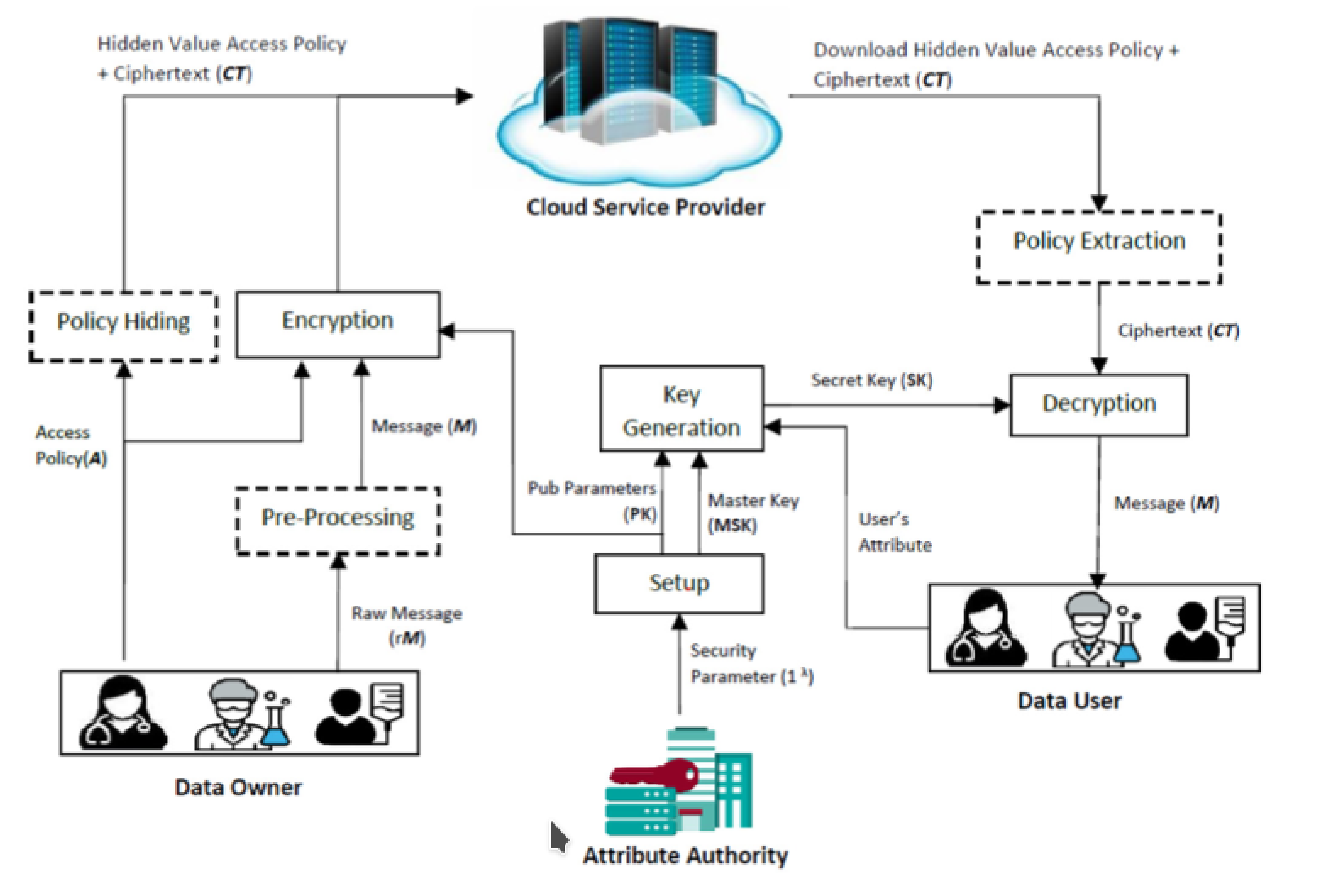
CIPHERTEXT-POLICY ATTRIBUTE-BASED ENCRYPTION

CP-ABE uses access trees to encrypt data and users' secret keys are generated over a set of attributes.

CP-ABE

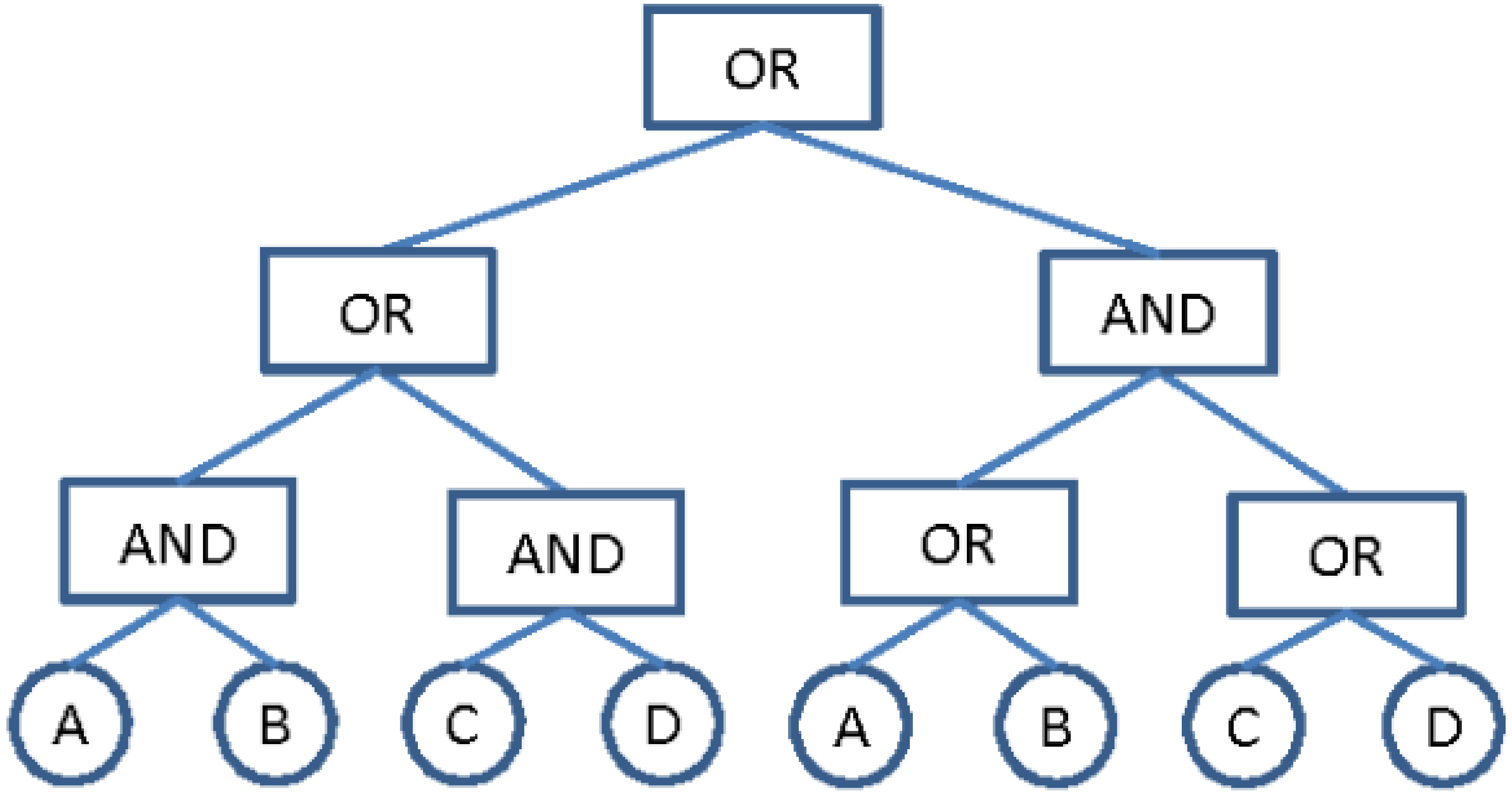
It has 5 main algorithms

- Setup
- Encrypt(PK,M, A)
- Key Generation(MK,S)
- Decrypt(PK, CT, SK)
- Delegate(SK, S)

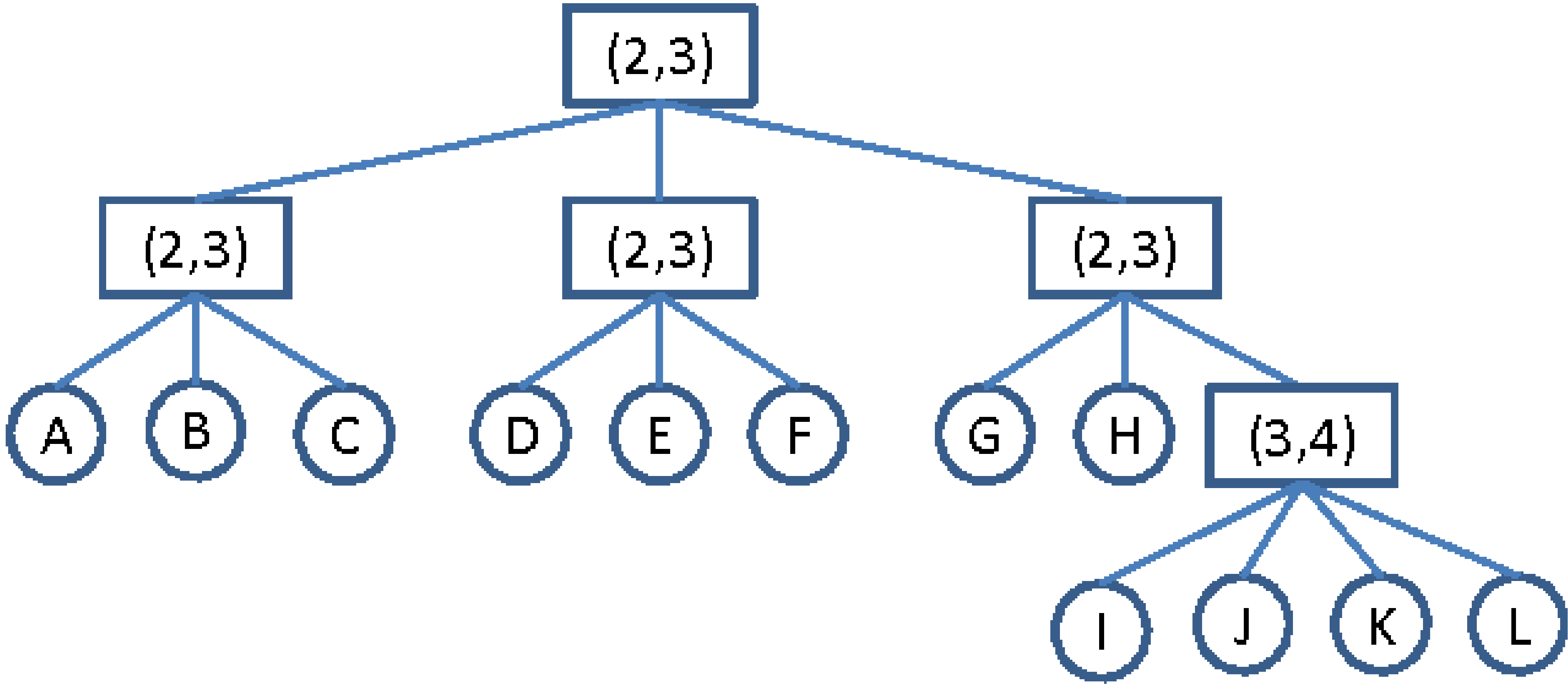


ACCESS TREE!

- Each non-leaf node of the tree represents a threshold gate.
- When the threshold is 1 it is an OR gate
- When Threshold is num(Children), it is an AND gate.
- only if x is a leaf node and denotes the attribute associated with the leaf node x in the tree.
- Satisfying an access tree ?



$$((A \wedge B) \vee (C \wedge D)) \vee ((A \vee B) \wedge (C \vee D))$$



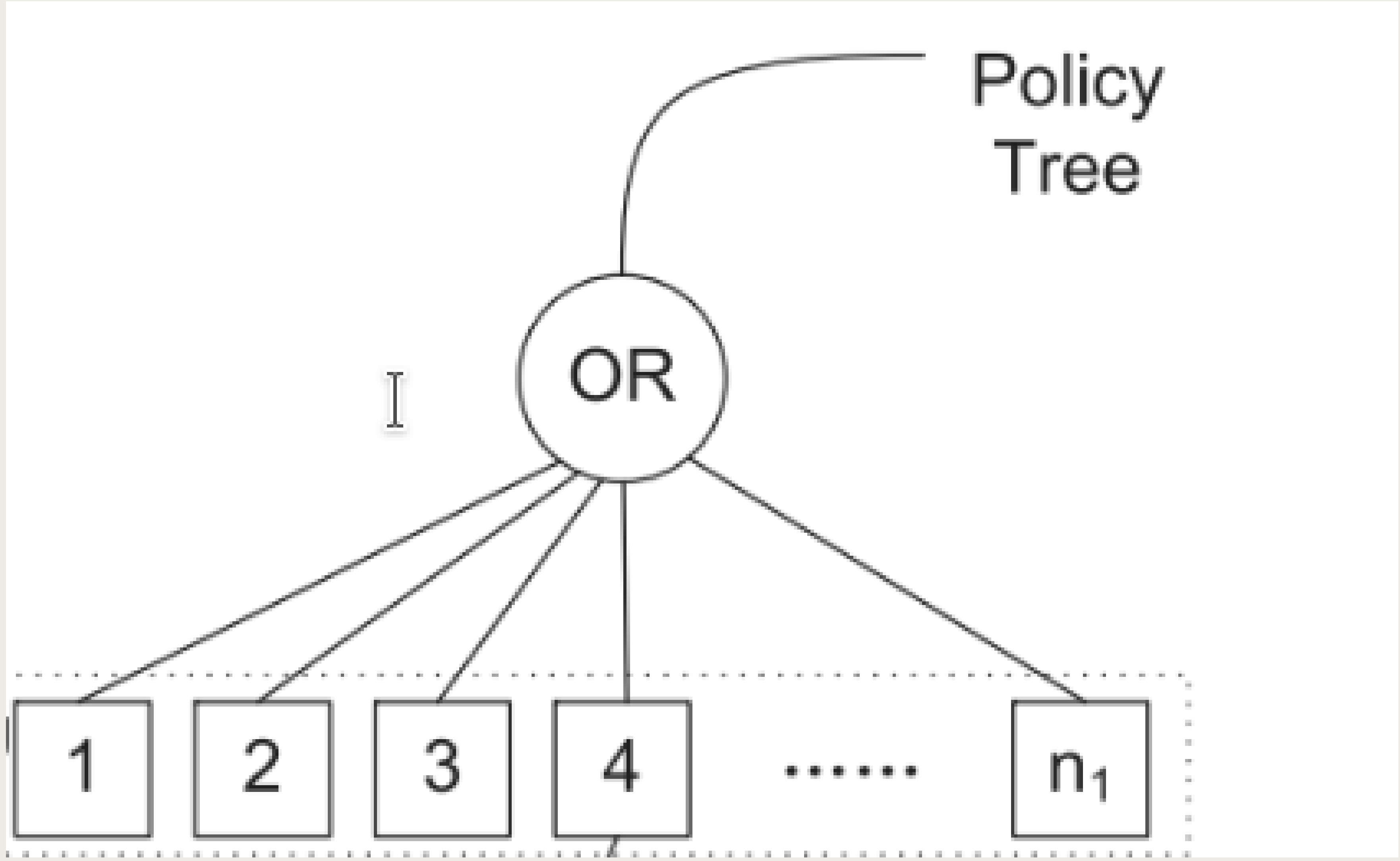
PROBLEM?

- Standard CP-ABE uses only AND, OR gate
- How to implement attribute based Comparsion on the Existing implementation using AND, OR gate
- example “(Distance < 1000 miles) AND (Date > May 1st)”.
- is Efficient in both space and Time complexity?

NAIVE APPROACH

For Example:-
“Distance < 1000 miles”

can be represented as a formula like
(“Distance = 999 miles” _ “Distance = 998 miles” _ ... _ “Distance = 0 miles”),
but the overhead increases linearly with the growth of attribute’s value space, which will become a performance bottleneck of the system.



WAY TO OPTIMAL APPROACH

The 0-encoding of s is defined as a set S0s such that

$$S_s^0 = \{s_n s_{n-1} \ldots s_{i+1} 1 | s_i = 0, 1 \leq i \leq n\}.$$

The 1-encoding of s is the set S1s such that S1s

$$S_s^1 = \{s_n s_{n-1} \ldots s_i | s_i = 1, 1 \leq i \leq n\}.$$

0-Encoding and 1-Encoding of 11 and 6

	1-encoding	0-encoding
$x=1011_2$	1 101 1011	11
$y=0110_2$	01 011	1 0111

here snsn-1sn-2.....s0 is a bit representation of N

OPTIMAL APPROACH

using this approach is way more efficient
Space Efficiency: maximum log N leaf nodes are added
Time Efficiency: instead of comparing N leaf nodes attributes we now only compare log N leaf nodes

Because the maxium number of 0/1 encoding's are log N

$$x > y \iff S_x^1 \bigcap S_y^0 \neq \varnothing.$$

0-Encoding and 1-Encoding of 11 and 6

	1-encoding	0-encoding
x=1011_2	1 101 1011	11
y=0110_2	01 011	1 0111

HOW TO ADD THESE IN ACCESS TREE?

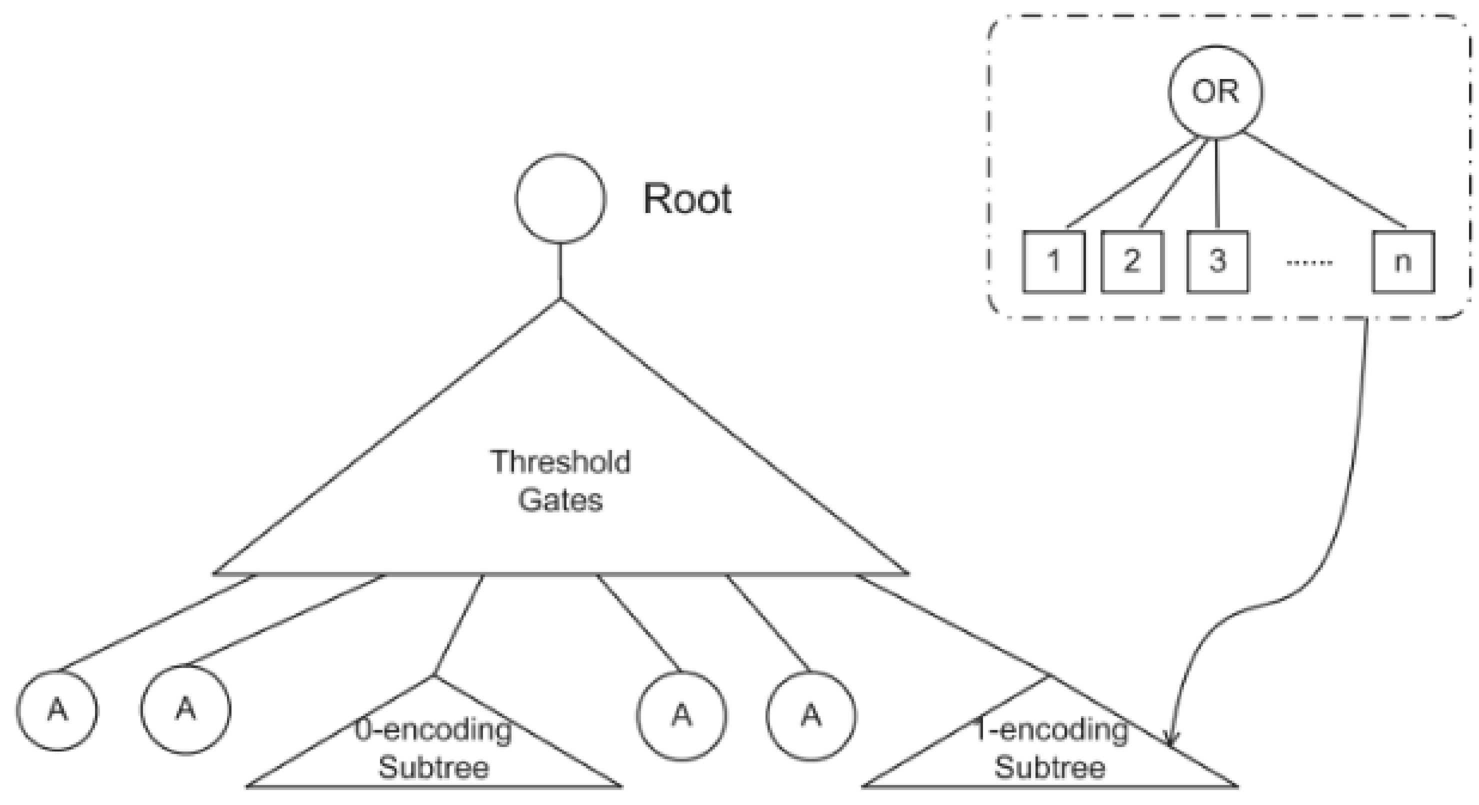
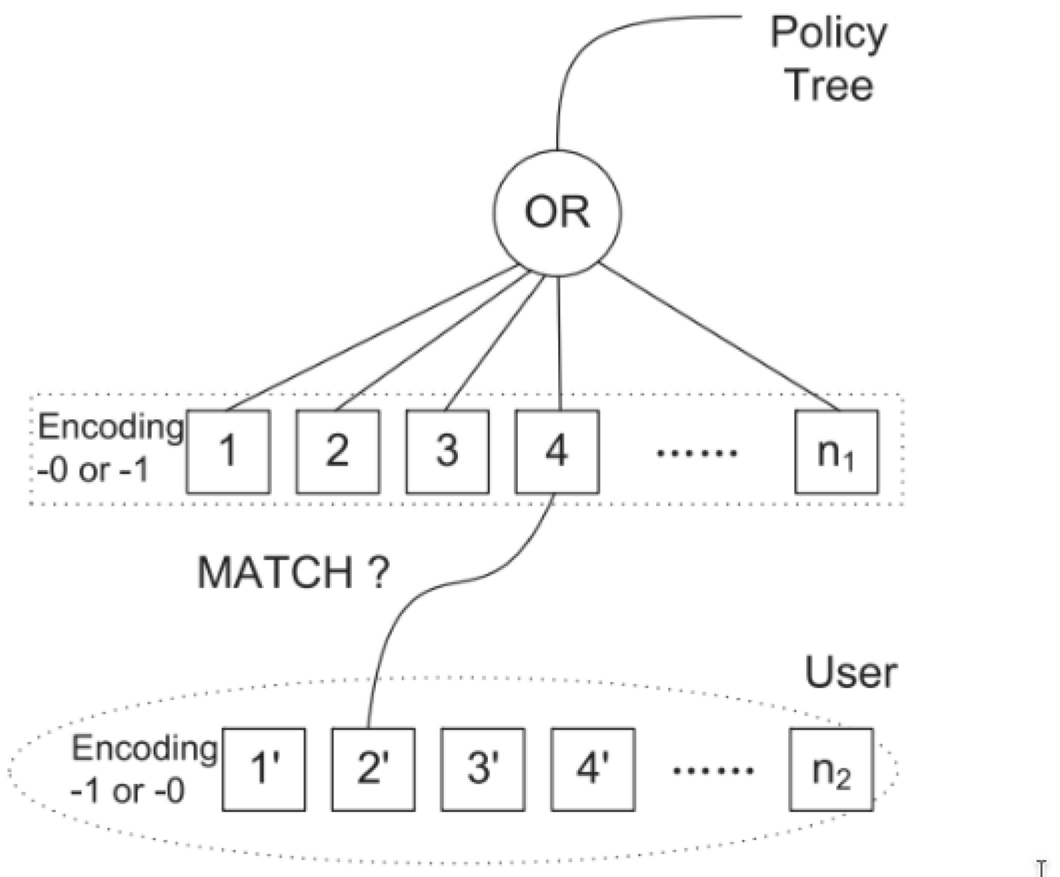


Fig. 2. Model of access structure with comparable attributes.

So as shown in the figure if the access structure has
if x<a then 1 encoding of a is added to existing tree
if x>a then 0-encoding of a is added to existing tree

the leaves in the subtree are
if x>a
<ATTRIBUTE_NAME || ">a" || ith(0)Encoding>
where || means concatinating



What for

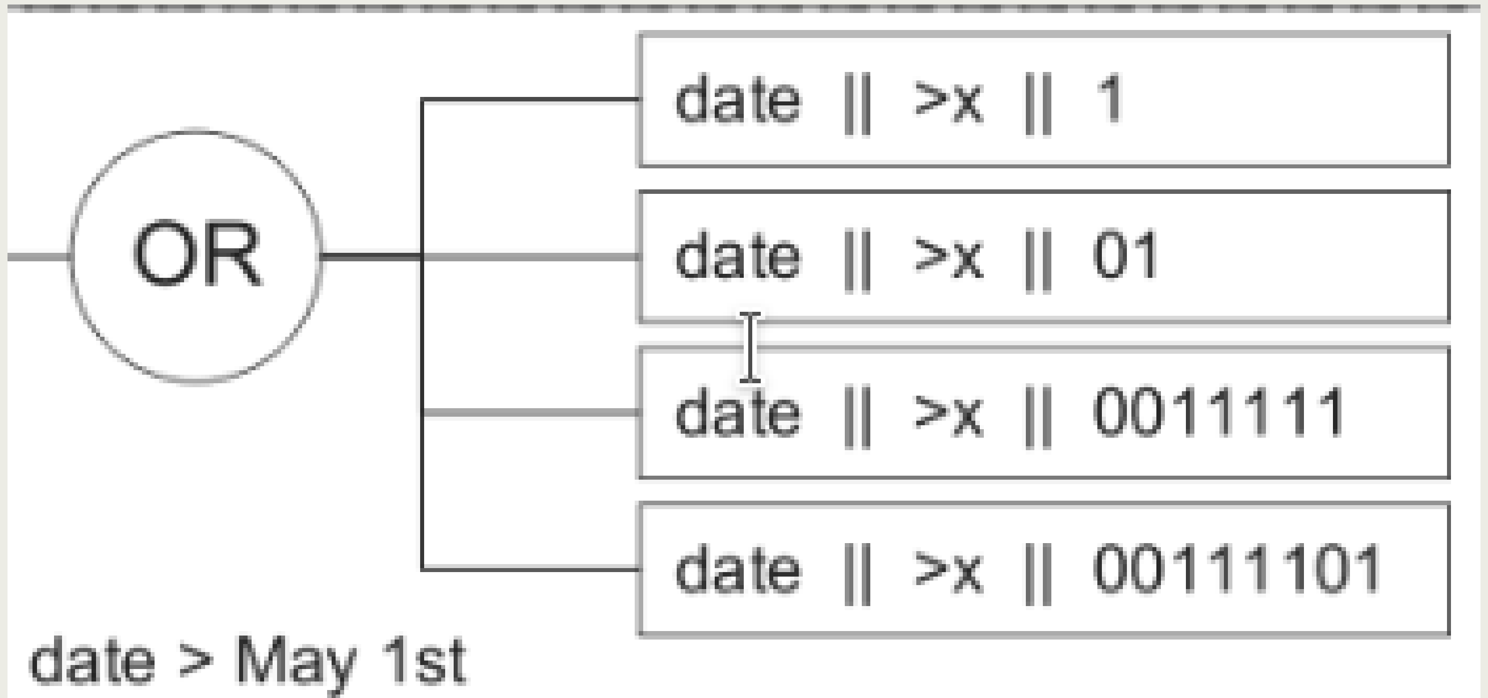
Negative Values

If the values are negative you can just change all the values by Xi = Xi-Xmin so that all the values are greater than equal zero

Now we can create a zero and one encoding for them

Other Ranges like Dates?

“(Date > May 1st)”
“May 1st” is expressed as an integer “121”(Which means the 121st day of a year.). Then we generate 1000’s 1-encoding and 121’s 0-encoding.



0/1 Encoding

Implemented a encode_number function responsible for generating the 0 and 1 encoding for a given number which was used further to perform set intersection and hence do range comparison

The time complexity for this function is log(n) as it just iterated over the bits of a give number which are log(n) are in number. The formula for 0 and 1 encoding are given previously.

```python
def encode_number(x):
    # Convert the integer to a binary string
    binary_string = bin(x)[2:]

    # Pad the binary string with leading zeros if needed
    # standardize all binary strings to be of length 32
    binary_string = binary_string.zfill(32)
    # Initialize 0-encoding and 1-encoding sets
    S0_x = set()
    S1_x = set()

    # Iterate over the binary string and populate the sets
    for i in range(len(binary_string)):
        prefix = binary_string[:i+1]
        if prefix[-1] == '0':

# Flip the least significant bit when adding to S0_x
            S0_x.add(prefix[:-1] + '1')
        else:
            S1_x.add(prefix)
    # convert the binary to decimal
    # sort S0_x and S1_x by length of the binary string
    S0_x = sorted(S0_x, key=lambda x: len(x), reverse=True)
    S1_x = sorted(S1_x, key=lambda x: len(x), reverse=True)
    return S0_x, S1_x
```

Access Policy Modification for Range Comparison

The modify_access_policy function is responsible for modifying the numerical range comparisons in the given access policy such that they are encoded via their 0 or 1 encoding form as per the requirement of comparison and the access policy string is updated automatically using that. This modified access_policy is fed to the access structure and when the modified attributes for the user is fed to the access structure it returns the boolean value accordingly

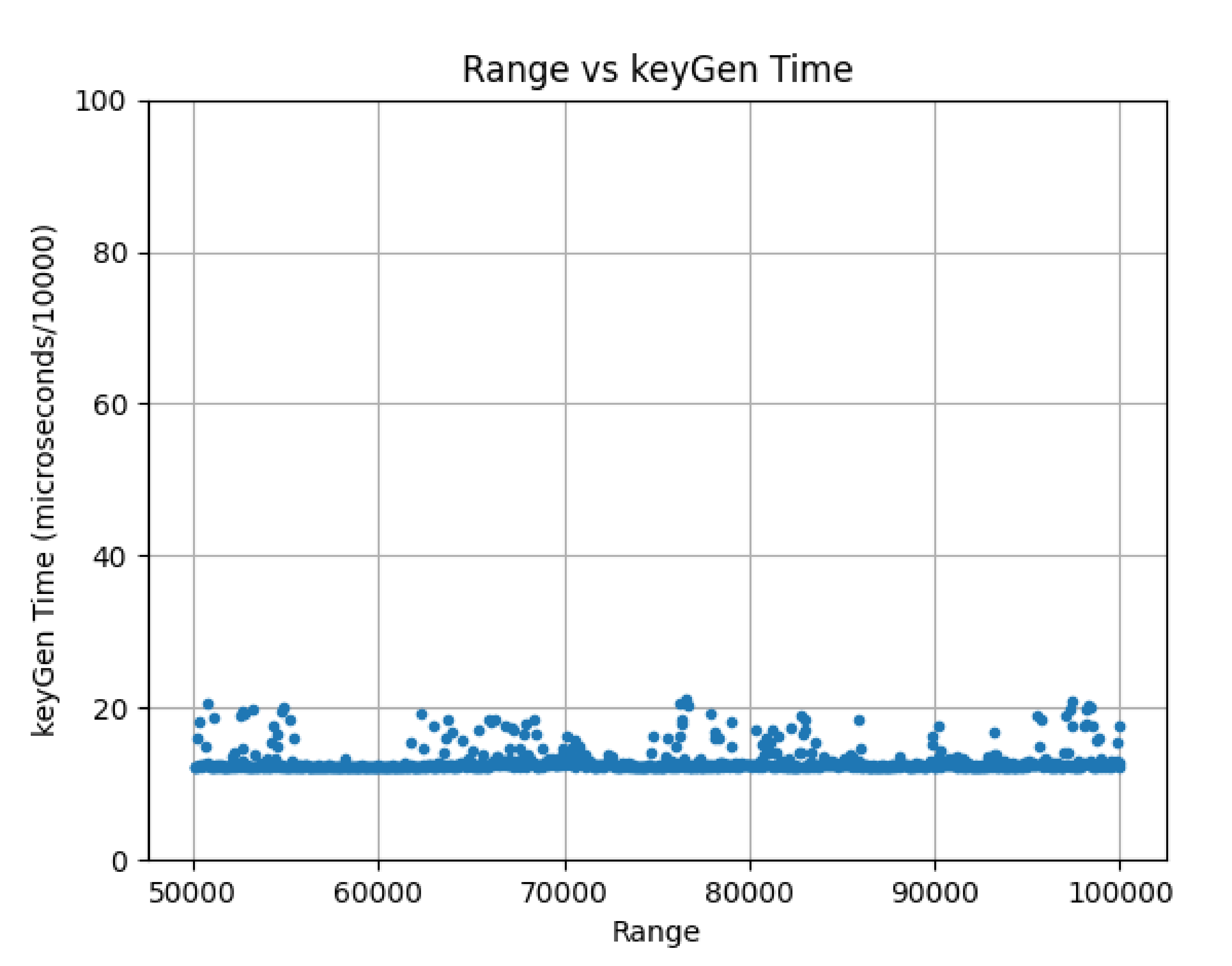
```
def modify_access_policy(access_policy):
    access_policy = modify_not_equal_conditions(access_policy
)
    pattern = re.compile(r'(\w+)\s*([<>])\s*(\d+)')
    matches = pattern.findall(access_policy)
    for match in matches:
        identifier, operator, number = match
        access_policy = access_policy.replace(
            f'{identifier} {operator} {number}', f'({
identifier} {operator} {number})')
    matches = pattern.findall(access_policy)
    for match in matches:
        identifier, operator, number = match
        S0_x, S1_x = encode_number(int(number))
        if operator == '<':
            new_condition = ' OR '.join(f'{identifier}{"!!"}{
x}' for x in S1_x)
        elif operator == '>':  # operator == '>'
            new_condition = ' OR '.join(f'{identifier}{"@@"}{
x}' for x in S0_x)
        access_policy = access_policy.replace(
            f'({identifier} {operator} {number})', f'({
new_condition})')
    pattern = re.compile(r'(\w+) = (\w+)')
    modified_policy = access_policy
    for match in pattern.findall(access_policy):
        old_expression = f'{match[0]} = {match[1]}'
        new_expression = f'({match[0]}$${match[1]})'
        new_expressionx = new_expression.upper()
        modified_policy = modified_policy.replace(
            old_expression, new_expressionx)
    return modified_policy
```

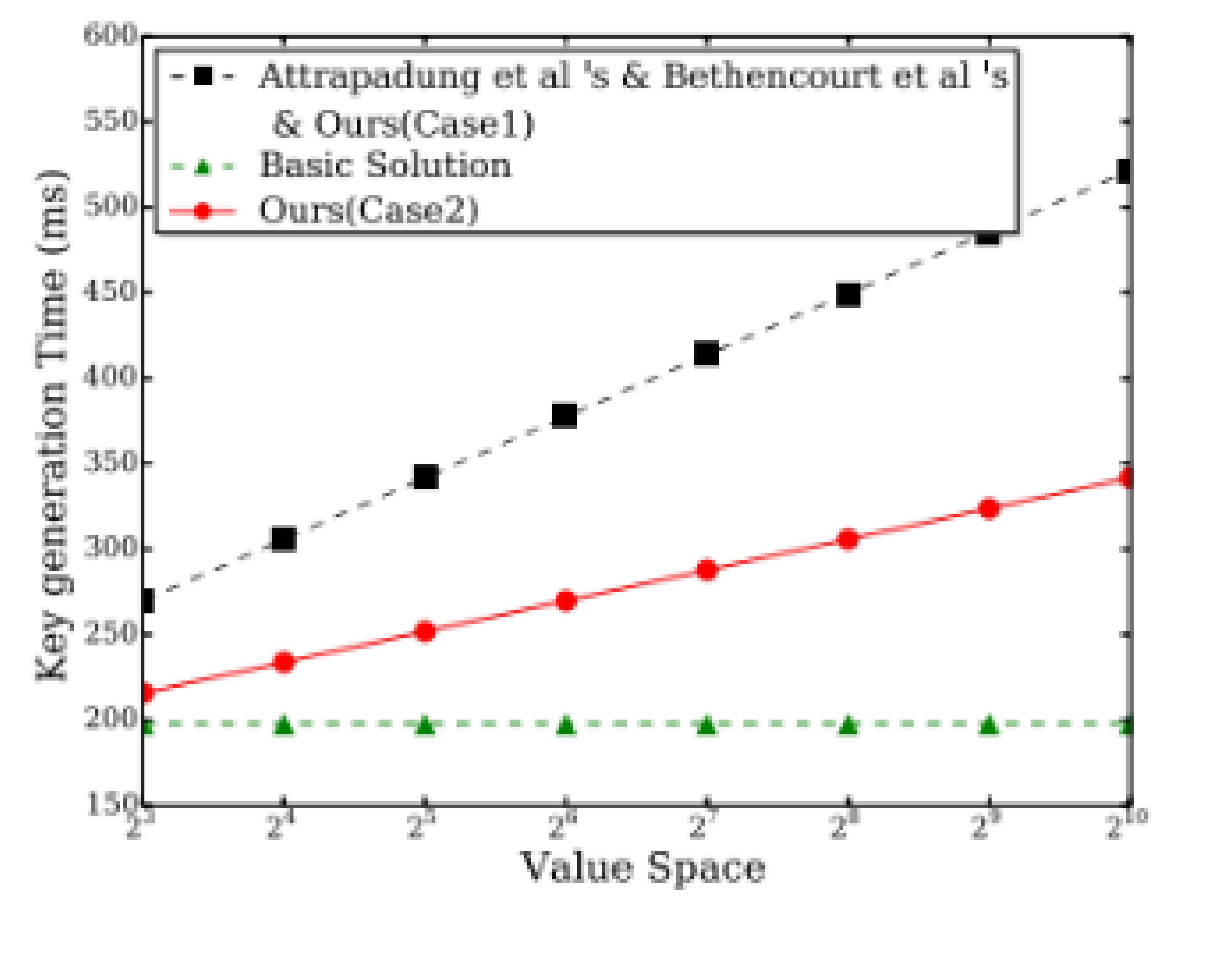
TESTING & BENCHMARKING

```
def modify_access_policyx(access_policy, num_attributes):
    # Split the original access policy string into individual attributes
    attributes = access_policy.split()

    # Create a list to store the modified attributes
    modified_attributes = []
    for i in range(num_attributes):
        # Generate a random attribute and append it to the list
        modified_attributes.append(generate_random_attribute(i))

        # Randomly choose between AND and OR and append it to the list
        if random.choice([True, False]):
            conjunction = 'and'
        else:
            conjunction = 'or'
        modified_attributes.append(conjunction)

    # Remove the last AND or OR if present
    if modified_attributes and (modified_attributes[-1] == 'and' or
modified_attributes[-1] == 'or'):
        modified_attributes.pop()

    # Join the modified attributes into a new access policy string
    new_access_policy = ' '.join(modified_attributes)
    new_access_policy = f"({new_access_policy})"

    return new_access_policy
```

ANALYSIS FOR KEYGEN vs Range Space

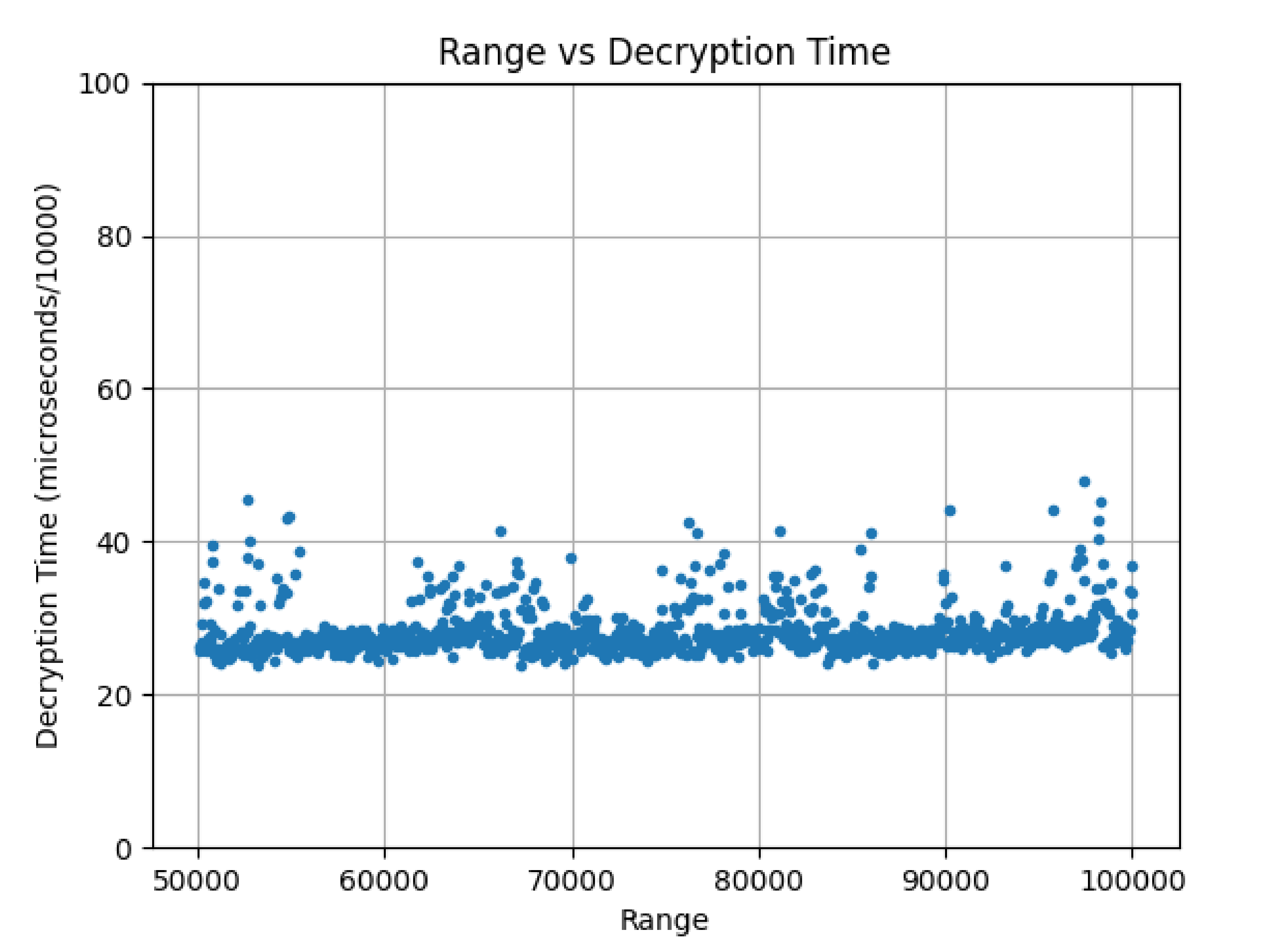
Range is |a-b| if access structure has x<a AND x>b

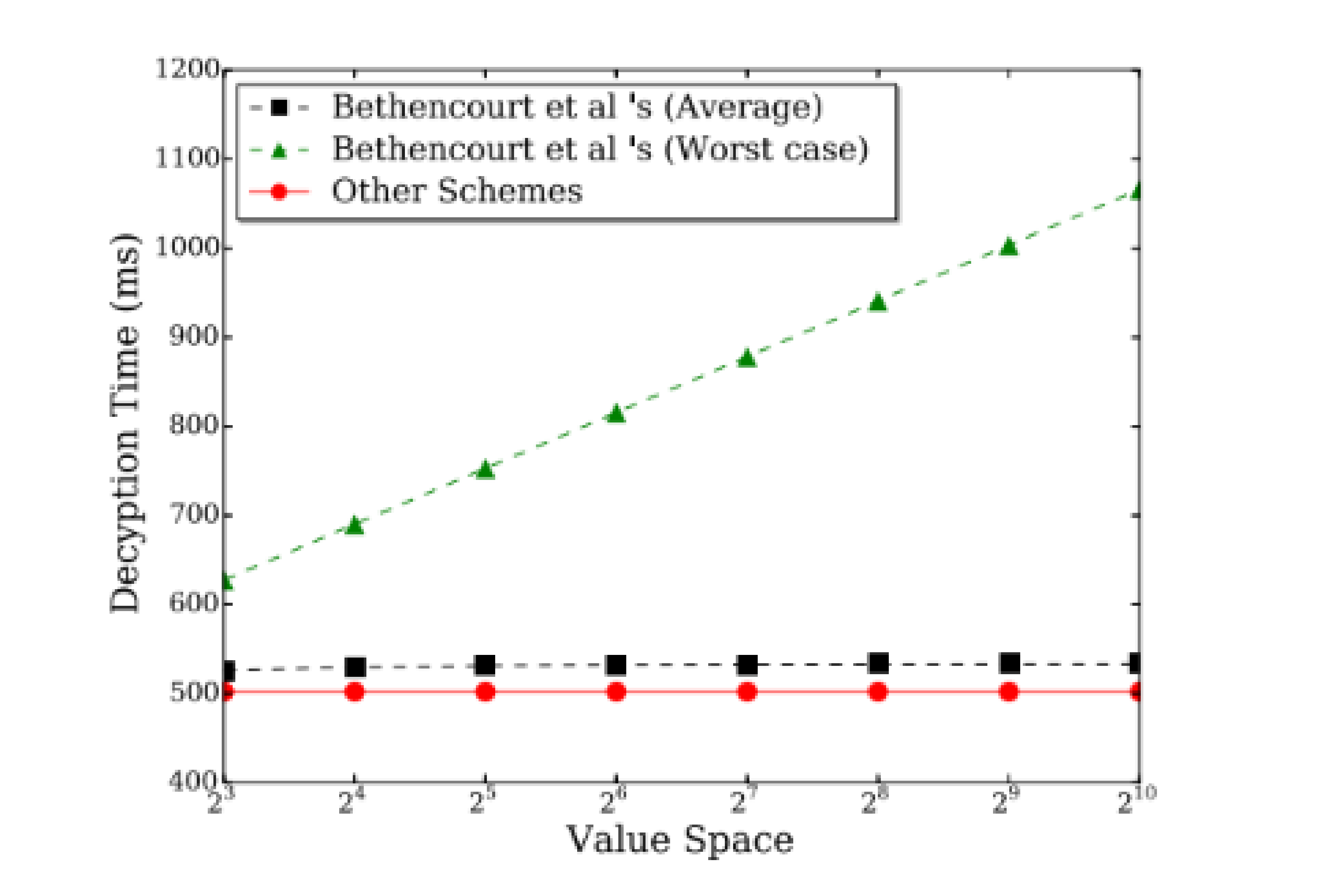




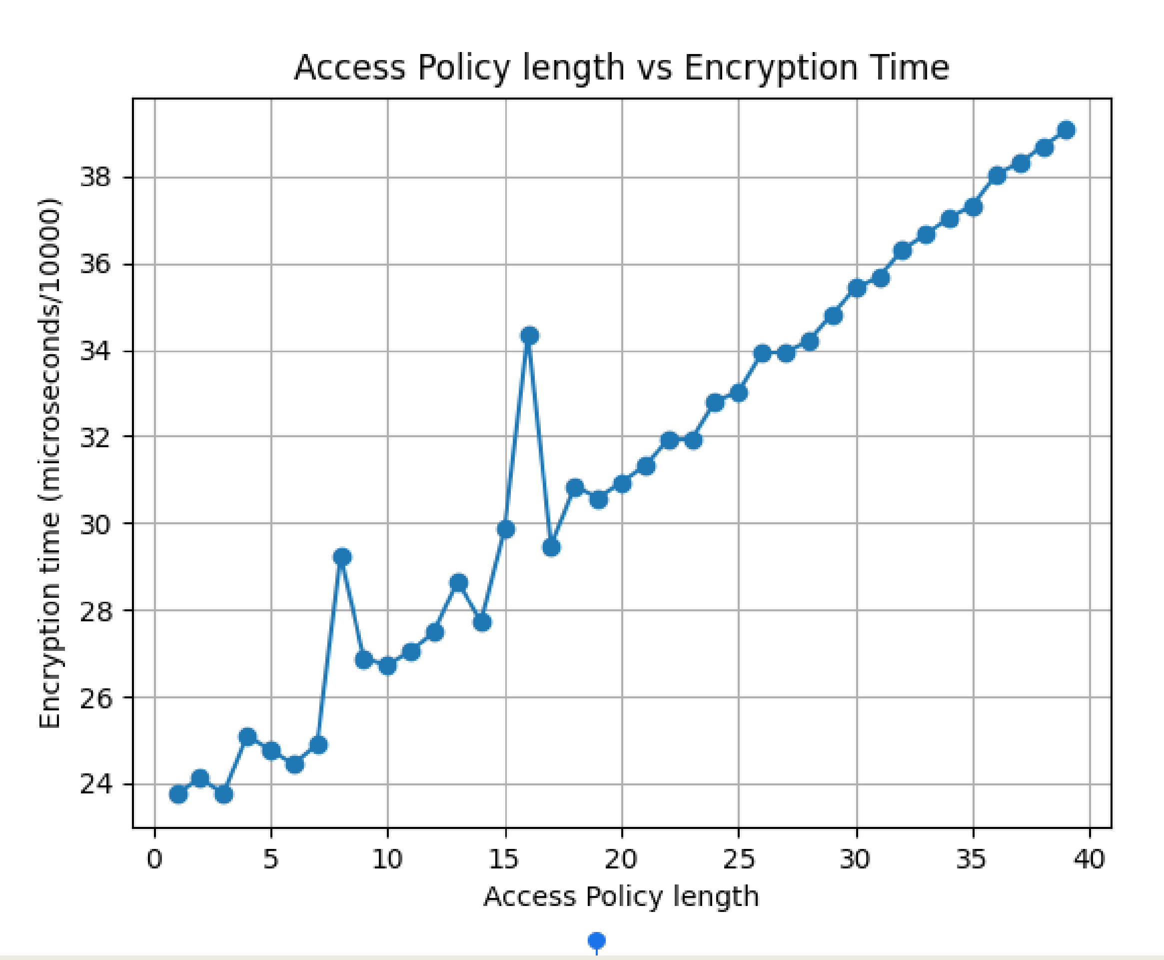
ANALYSIS FOR DECRYPTION vs Range Space

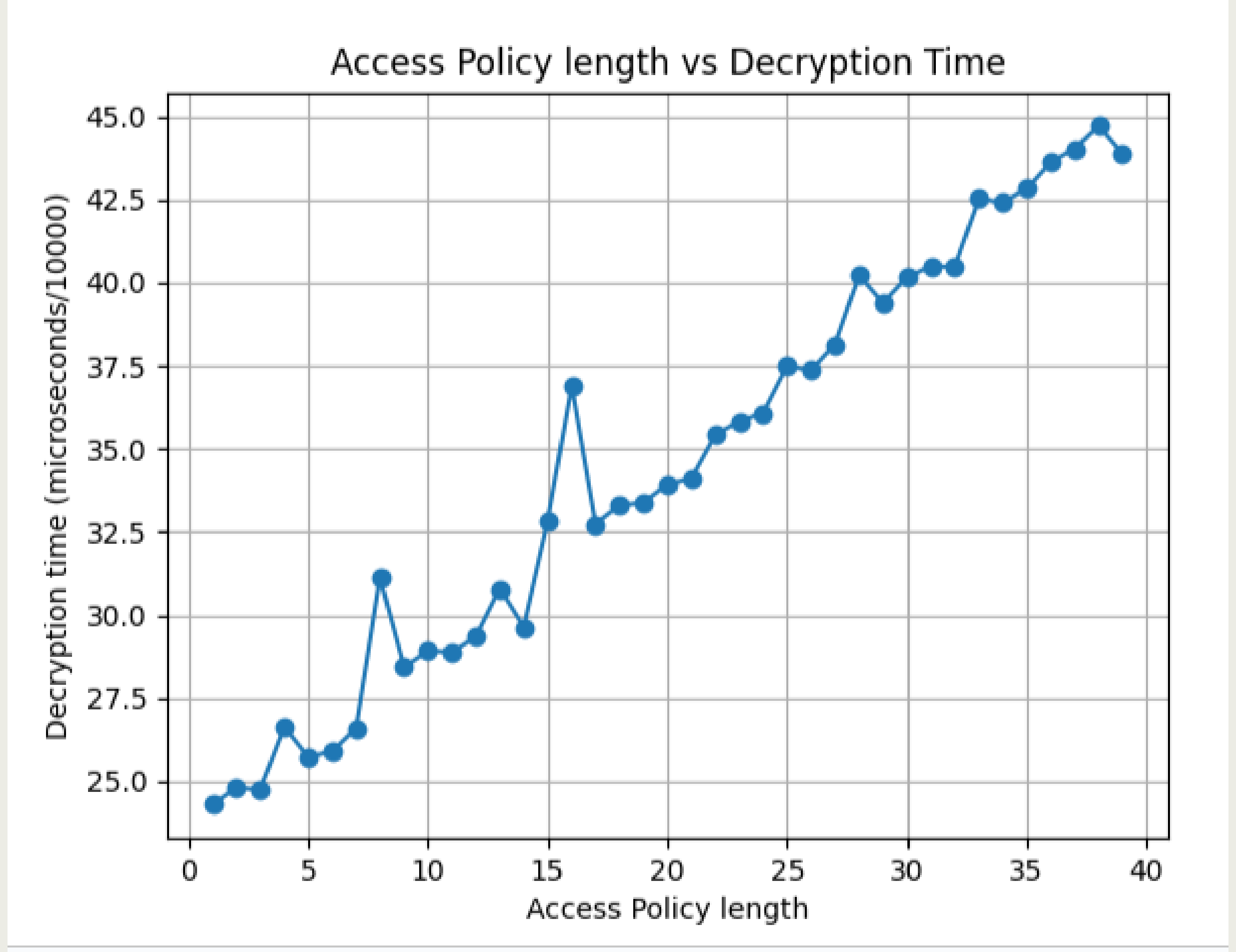
Range is |a-b| if access structure has x<a AND x>b

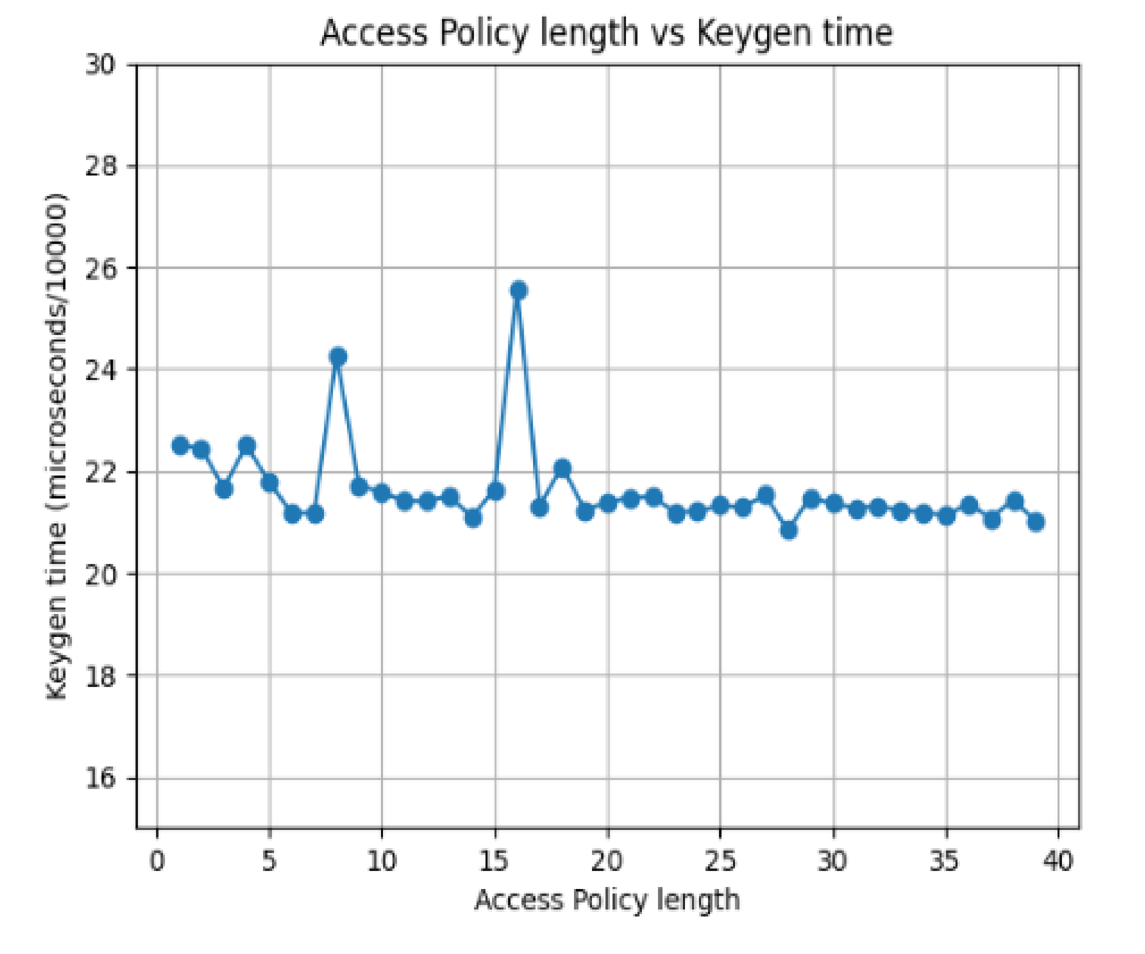




ANALYSIS FOR ALL vs no of Access policy







THEORETICAL DERIVATIONS

Performance Analysis Result with Theoretic Derivation

Overhead(AVG)	Basic Solution	Bethencourt et al.'s	Attrapadung et al.'s	Ours (Case1)	Ours (Case2)
Extended Attributes in Access Policy	$N/2$	$\log_2 N + \frac{\log_2 N - N + 1}{N}$	$N/2$	$\log_2 N/2$	
Extended Thresholds in Access Policy	1	$(\log_2 N - 1)/2$	$1 - \log_2 N/N$	$1 - \log_2 N/N$	
Security Key Extension	1	$\log_2 N$	$\log_2 N$	$\log_2 N$	$\log_2 N/2$

The object in this table is the average expanded overhead for an attribute field with value space N (We assume that $N = 2^n$, $n \in N^*$).

REFERENCES

1. CABE: A New Comparable Attribute-Based Encryption Construction with 0-Encoding and 1-Encoding (Kaiping Xue, Senior Member, IEEE, Jianan Hong, Yingjie Xue, David S. L. Wei, Senior Member, IEEE, Nenghai Yu, and Peilin Hong)
2. Efficient Encrypted Range Query on Cloud Platforms (PING YU, WEI NI, REN PING LIU, ZHAOXIN ZHANG , HUA ZHANG and QIAOYAN WEN)
3. Ciphertext-Policy Attribute-Based Encryption (Bethencourt , Amit Sahai , Brent Waters)

THANK YOU