

Searching in encrypted data

or "How your tax-evasion will only be seen by you! (and us)"

Tjeerd Boerman
Mattijs Ugen
Niels Visser
Peter Wagenaar

Agenda

1. Assignment
2. Requirements
3. Design
4. Implementation details
5. Demo
6. Conclusion & Questions

Assignment

- Consultant that stores (financial) data of his clients on a storage server
- Both he/she and the clients should be able to access and search in the data
- How to ensure data privacy when the server is honest but curious?

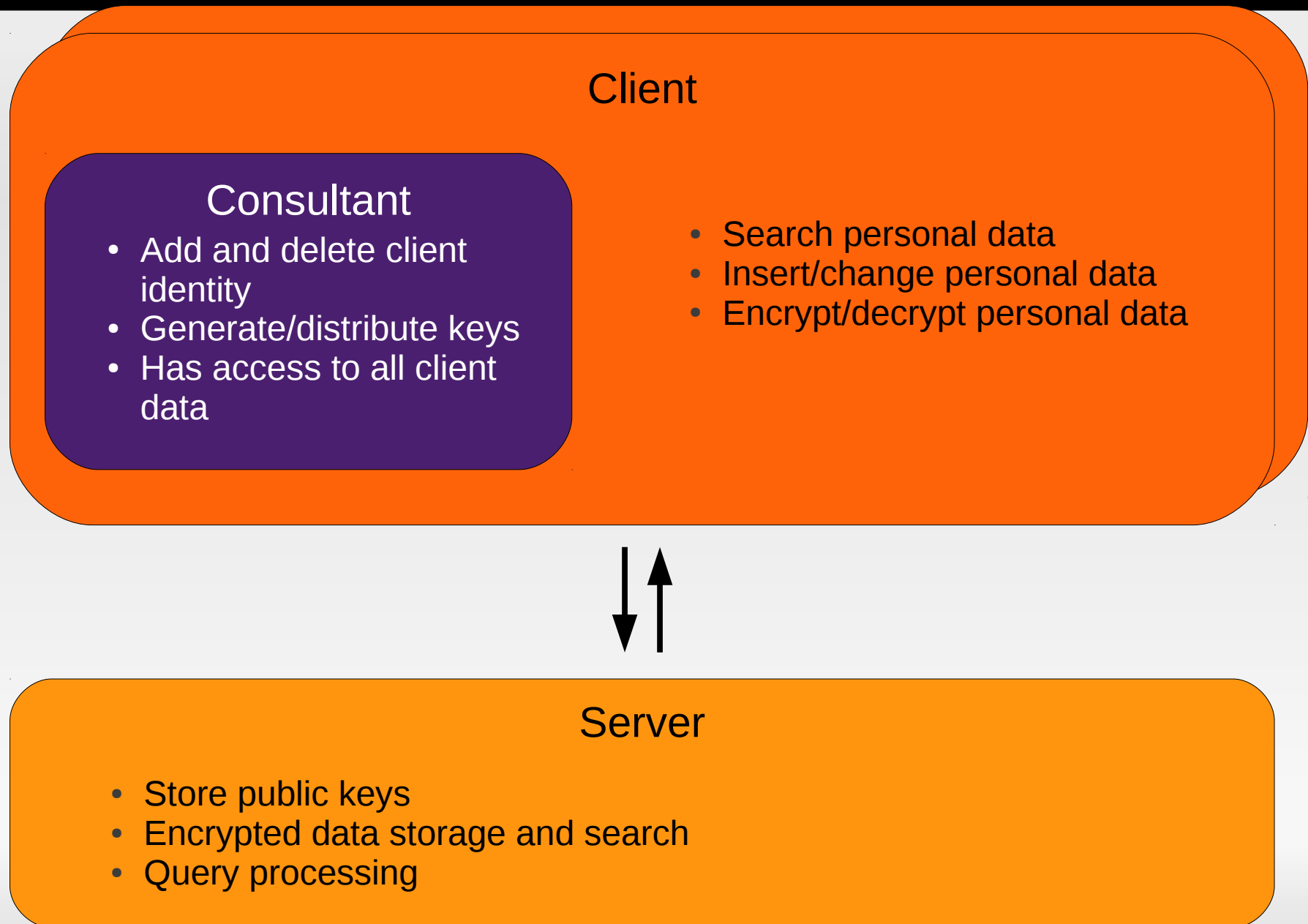


Requirements

- The consultant must have access to all data
- The clients may only access their own data
- Insertion and searching should be supported
- Storage server should not be able to learn about the actual data



Design



Design

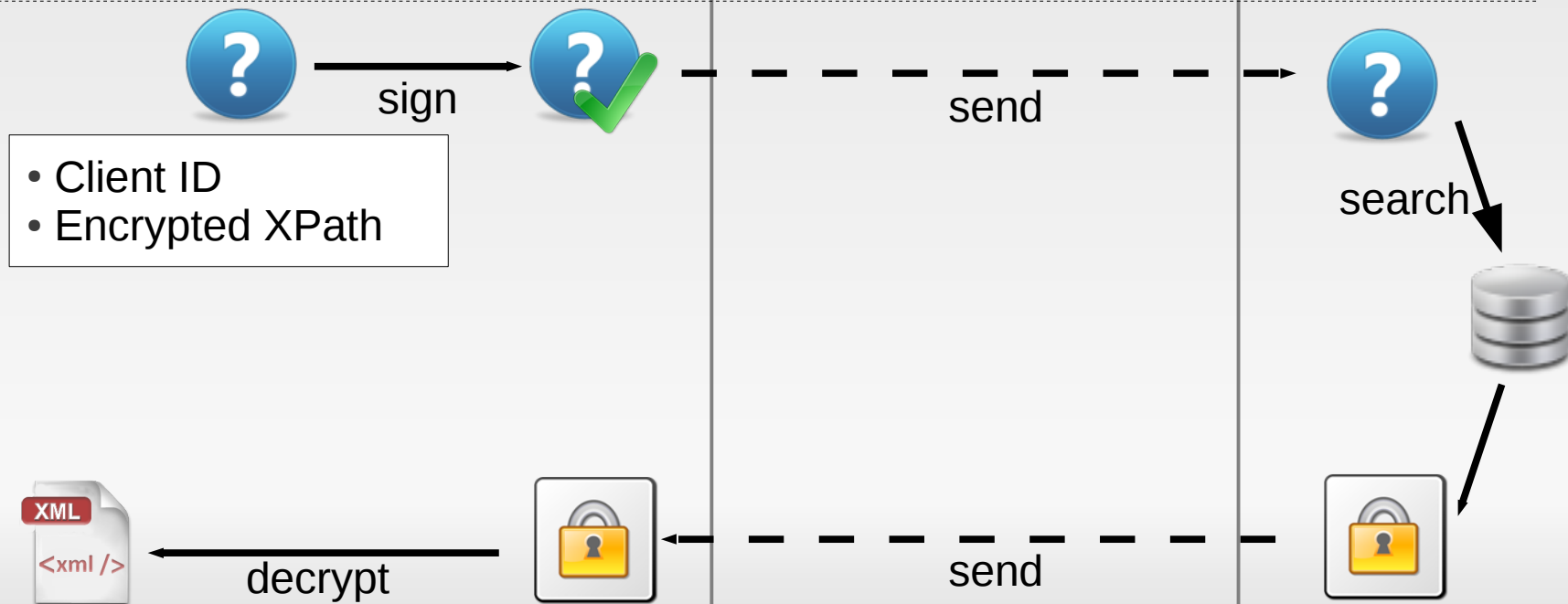
- Every client has three keys associated with his identity
 - Encryption key
 - Hash key
 - RSA private key
- The consultant has a private RSA key
- The server has all public keys

Design

Insert



Search:



Implementation Details

Client

Webapplication

PHP5

SQLite

Openssl

Mcrypt

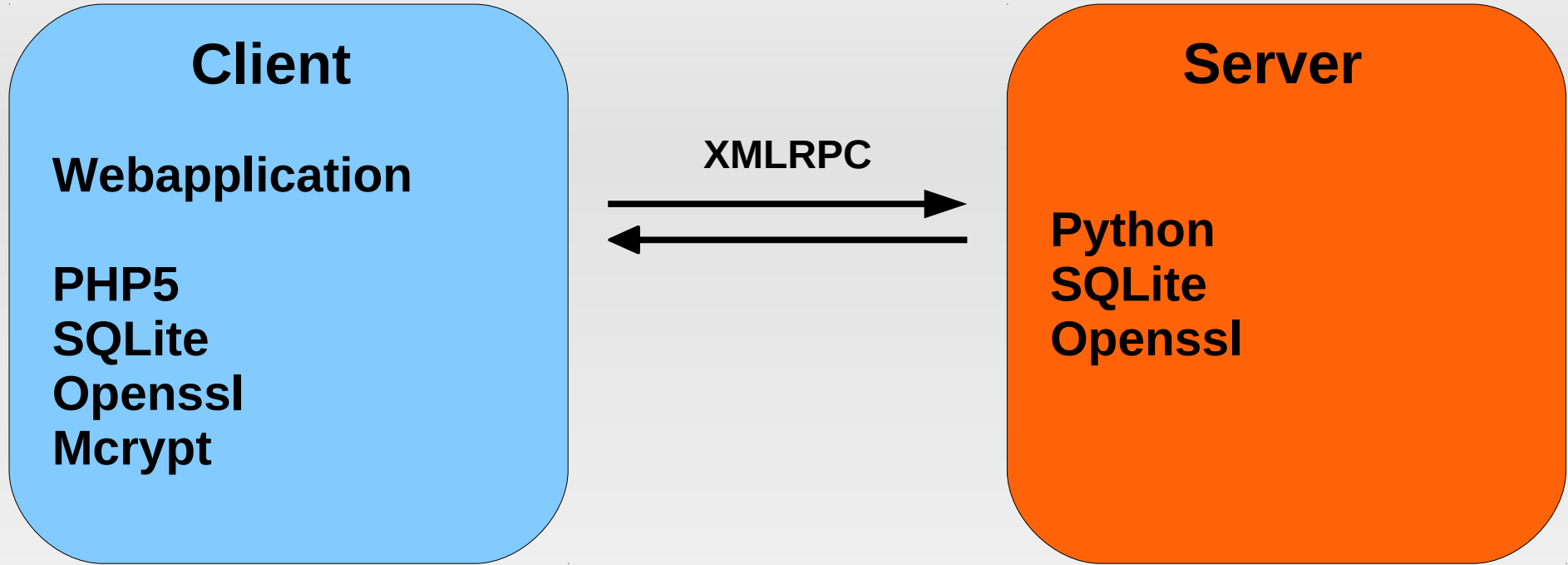
XMLRPC

Server

Python

SQLite

Openssl

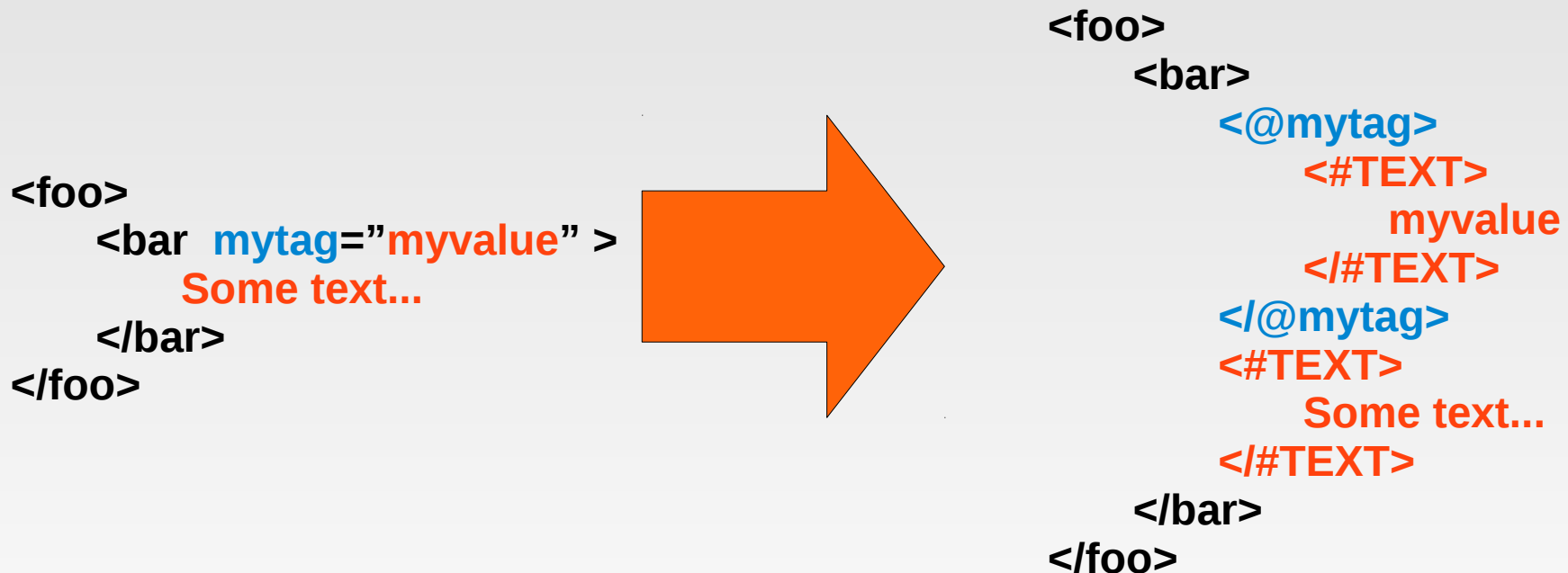


Implementation Details

- Searchable encryption approach from paper "Efficient Tree Search in Encrypted Data"
- XML document is converted so it can be stored in a relational database.
 - One row per XML node
- Each row is encrypted by the client according to the algorithm.
- Server can execute XPath queries on the data without learning about the XML tags or values.

Implementation Details

- Preprocess XML so everything is a node with a tag and value.



Implementation Details

- Convert XML nodes to pre-post-parent format.

	pre	post	parent
<foo>	0		-1
<bar>	1		0
<@mytag>	2		1
<#TEXT>	3		2
myvalue			
</#TEXT>		0	
</@mytag>		1	
<#TEXT>	4		1
Some text...			
</#TEXT>		2	
</bar>		3	
</foo>		4	

Implementation Details

- The resulting SQL rows then need to be encrypted by the client

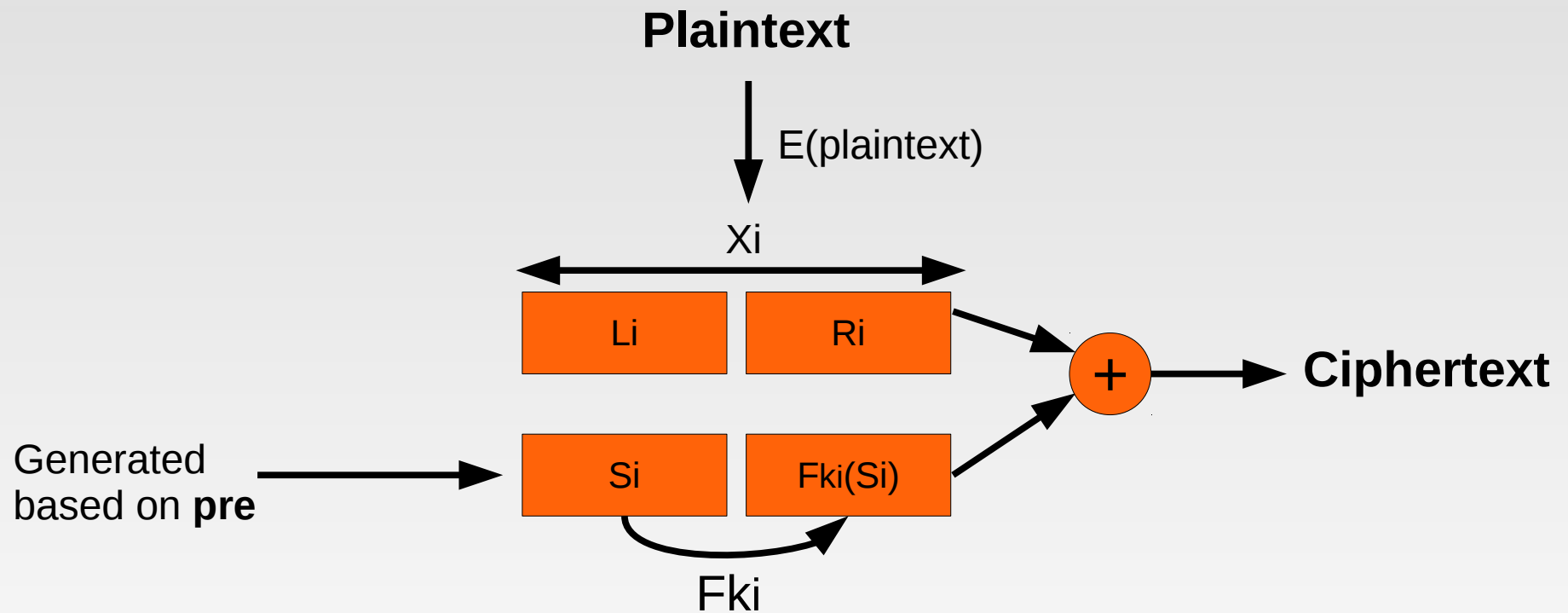
```
<foo>  
  <bar mytag="myvalue" >  
    Some text...  
  </bar>  
</foo>
```

pre	post	parent	tag	value
0	4	-1	foo	
1	3	0	bar	
2	1	1	@mytag	
3	0	2	#TEXT	myValue
4	2	1	#TEXT	Some text...

Encryption Scheme

- Recall: every client has three keys associated with his identity
 - Encryption key
 - Hash key
 - RSA private key

Encryption Scheme

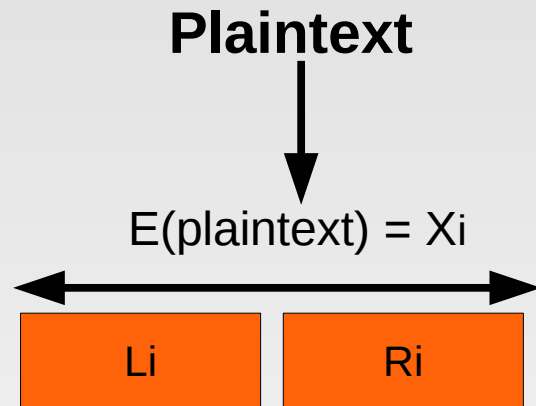


$E = 128$ bit AES

$F = 128$ bit AES

Encryption Scheme

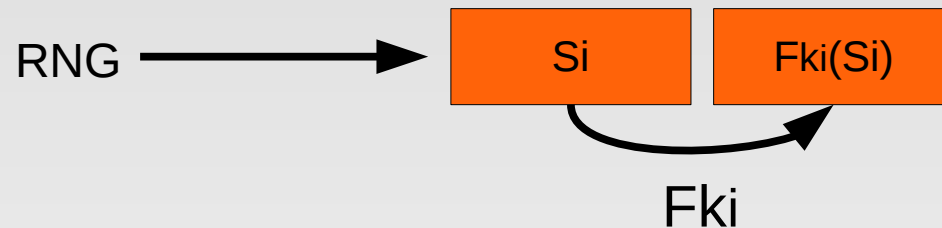
- Encryption key



- E is AES-128 keyed with the encryption key
- The encrypted plaintext is called X_i

Encryption Scheme

- Hash key



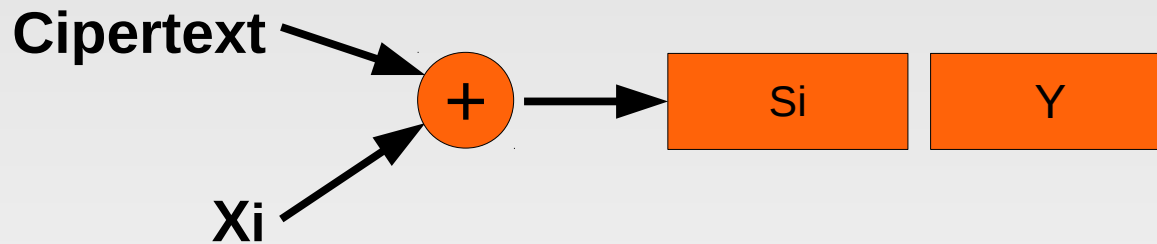
- F is AES-128 keyed with k_i
- k_i is the elements' unique hash key
- k_i is calculated by encrypting S_i with Hash key

Search Scheme

- XPath query: **/foo**
- Client reproduces $\langle k_i, X_i \rangle$ from the plaintext
- Plaintext terms in the Xpath query are substituted with these tuples.
- Query is sent to the server.

Search Scheme

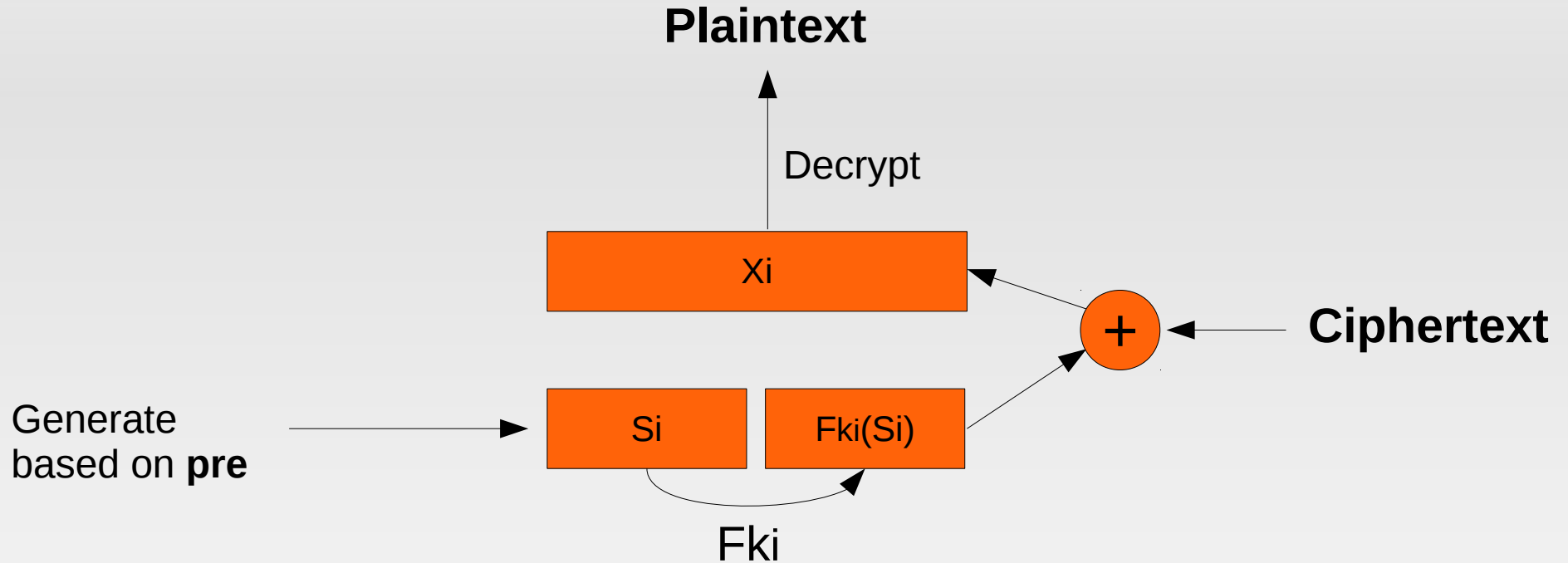
- The server checks the relation using the element-specific hash key k_i .



$$Y = F_{k_i}(S_i) \quad ?$$

- The node is a match when the hash matches Y .

Decryption Scheme



Demo

Conclusion & Questions

- Questions?

