How to Search on Encrypted Data

Eu-Jin Goh

Stanford University

19 November 2003

How to search on encrypted data?

Should we trust our remote storage?

Many reasons not to

1. Outsourced backups and storage
2. Sysadmins have root access
3. Hackers breaking in

Should we trust our remote storage?

Many reasons not to

1. Outsourced backups and storage
2. Sysadmins have root access
3. Hackers breaking in

Solution

1. untrusted file systems [GSMB03,MS02]
2. encrypt + MAC files before storing

Why we need efficient encrypted data search?

- Want all docs with “launch codes”
- But documents encrypted and server not trusted with contents nor key

Why we need efficient encrypted data search?

- Want all docs with “launch codes”
- But documents encrypted and server not trusted with contents nor key

Naïve Solution

- Download all documents, decrypt, and search on local machine
- Want “better” solutions than this

Design Goals

1. Minimize communication overhead
2. Minimize computation on both server and client
3. Multi-user setting
4. Practical – deployable right now

Security Wish List

From coded query, server cannot -

- distinguish between documents
- determine document contents
- see search keyword
- learn anything more than result

Server cannot generate coded query

3 Solutions

1) Practical Techniques for Searches on Encrypted Data (SSKE)

   D. Song, D. Wagner, and A. Perrig.

2) Searchable Public Key Encryption (SPKE)

   D. Boneh, G. Crescenzo, R. Ostrovsky, G. Persiano

3) Secure Indexes for Searching Efficiently on Encrypted Compressed Data

   E.-J. Goh

How to search on encrypted data?

Solution 1 -

Practical Techniques for Searches on Encrypted Data (SSKE)

D. Song, D. Wagner, and A. Perrig

Overview

- Focus on non-index solution
  - Sequential scan entire document
- Searchable Symmetric Key Encryption (SSKE)
- Index solution uses hash tables but updates insecure

Pseudo-Random Functions

Intuitively

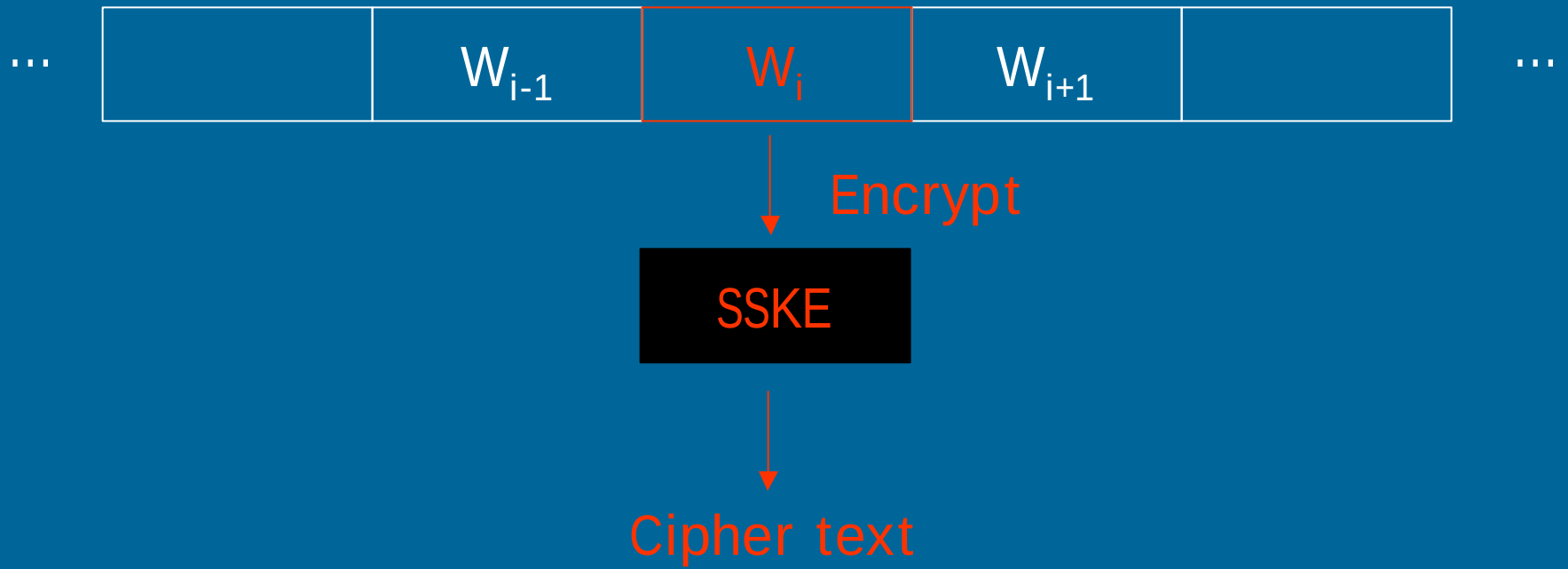
- PRF's indistinguishable from random functions
- Given $x_1,\ldots,x_m$ and $f_k(x_1),\ldots,f_k(x_m)$, adversary cannot predict $f_k(x_{m+1})$ for any x_{m+1}

Linear Search with SSKE

Treat document as a series of keyword blocks.

Linear Search with SSKE

To Encrypt



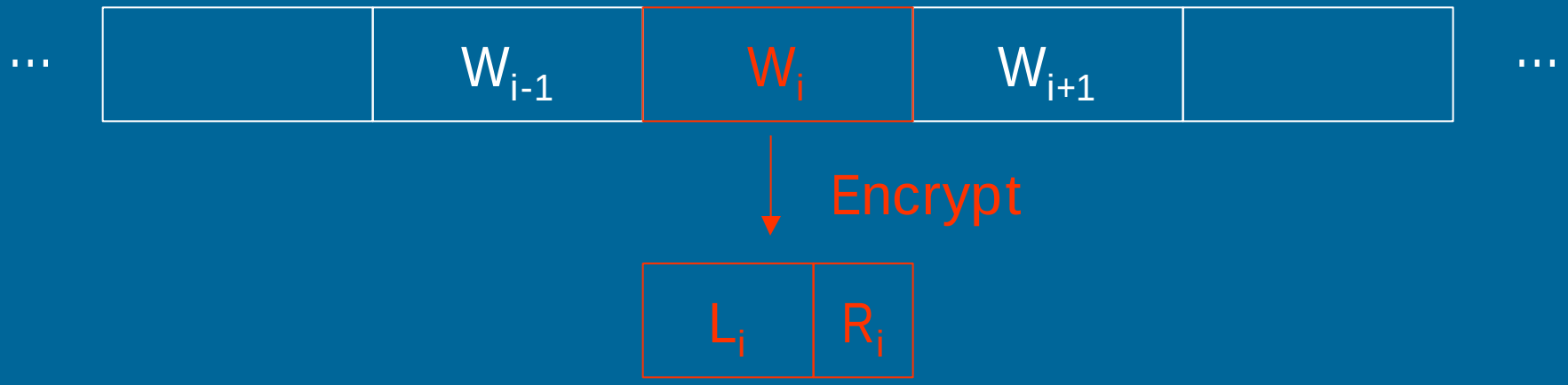
Searchable Symmetric Key Encryption (SSKE)

SSKE Operations

1. Key Generation
2. Encrypt
3. Generate Trapdoor
4. Test for keyword

SSKE - Encrypt

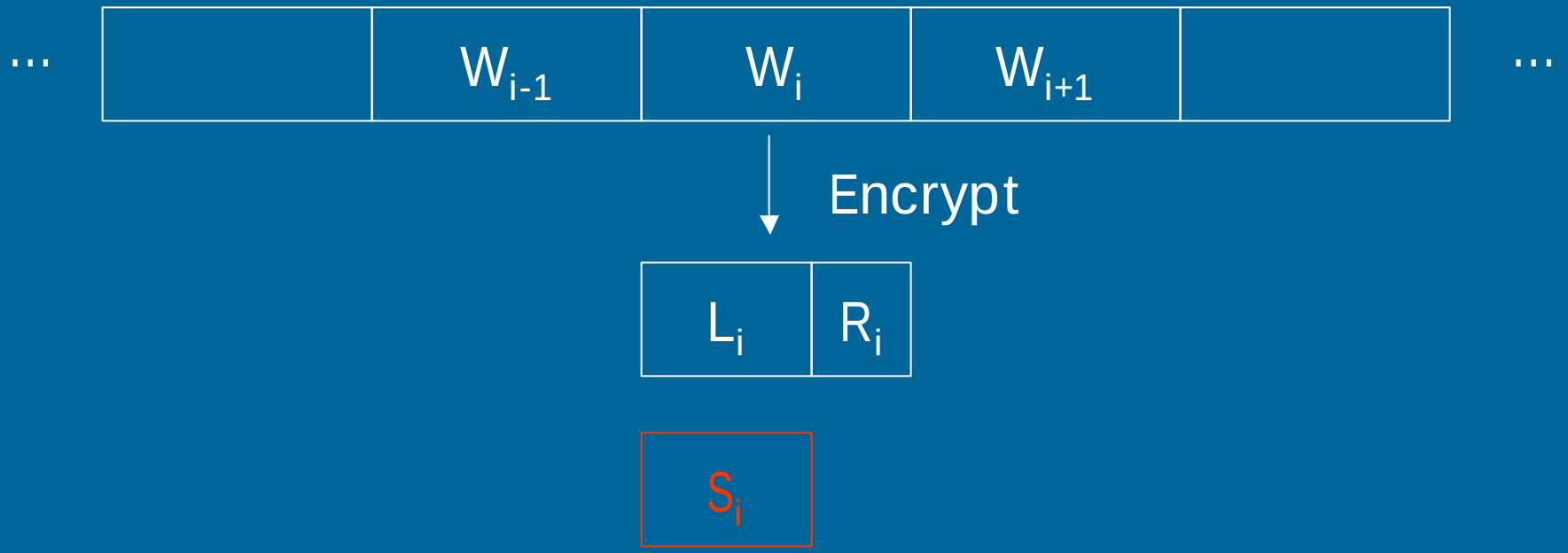
To Encrypt



1) Encrypt using a deterministic cipher with key a. Divide cipher text into 2 parts, L_i and R_i

SSKE - Encrypt

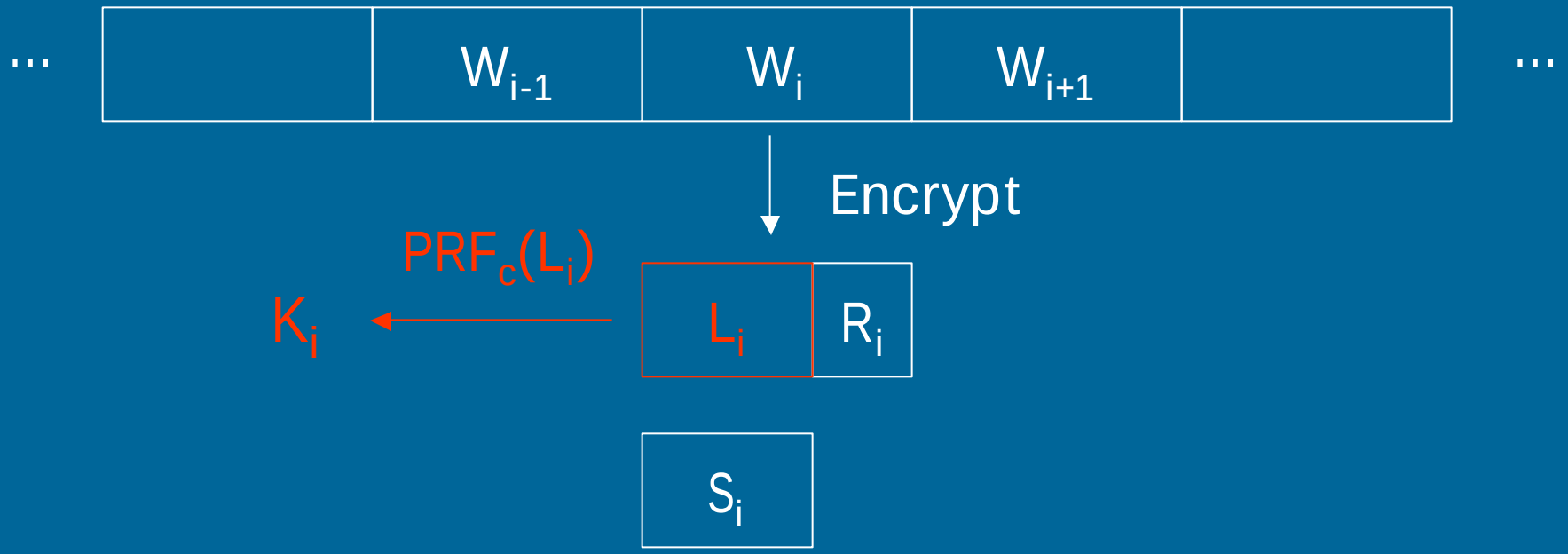
To Encrypt



2) Use a PRG with key b to generate random bits S_i based on the location of W_i

SSKE - Encrypt

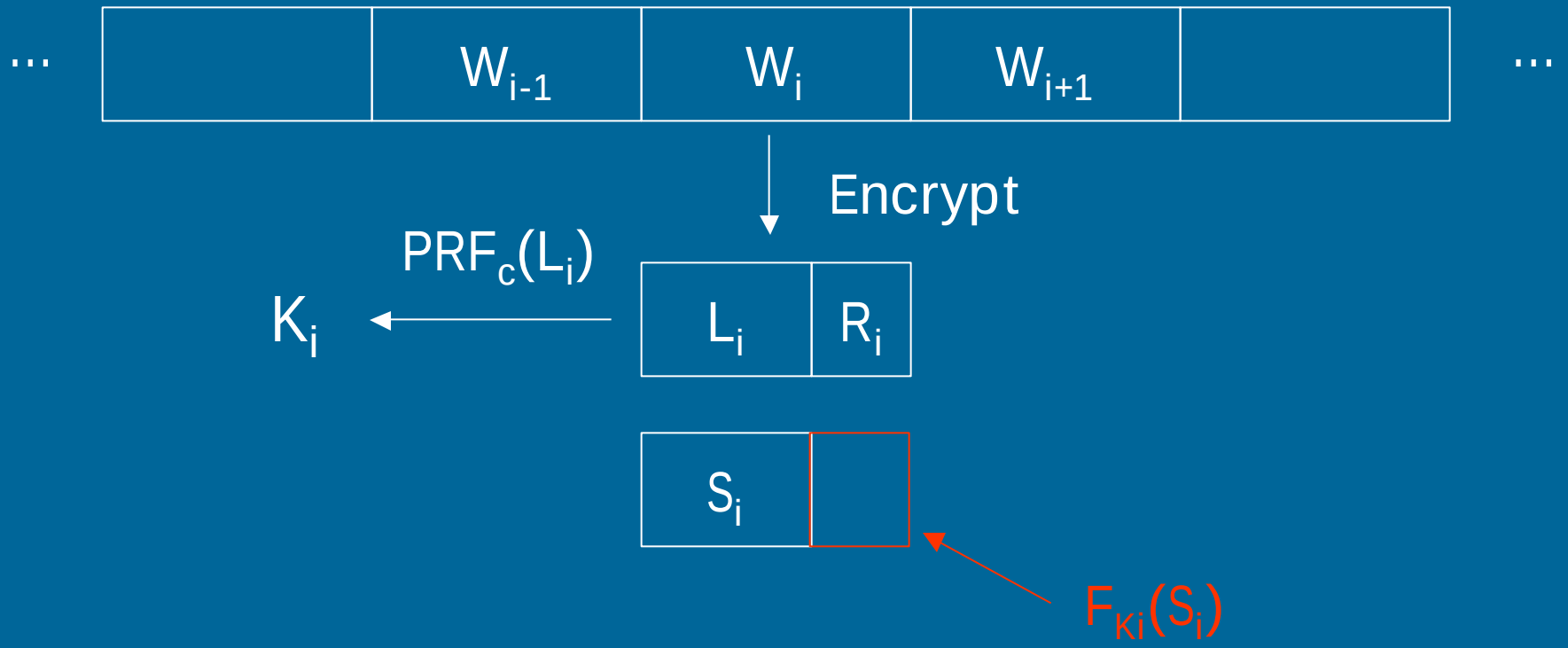
To Encrypt



3) Use a PRF with key c to derive key K_i

SSKE - Encrypt

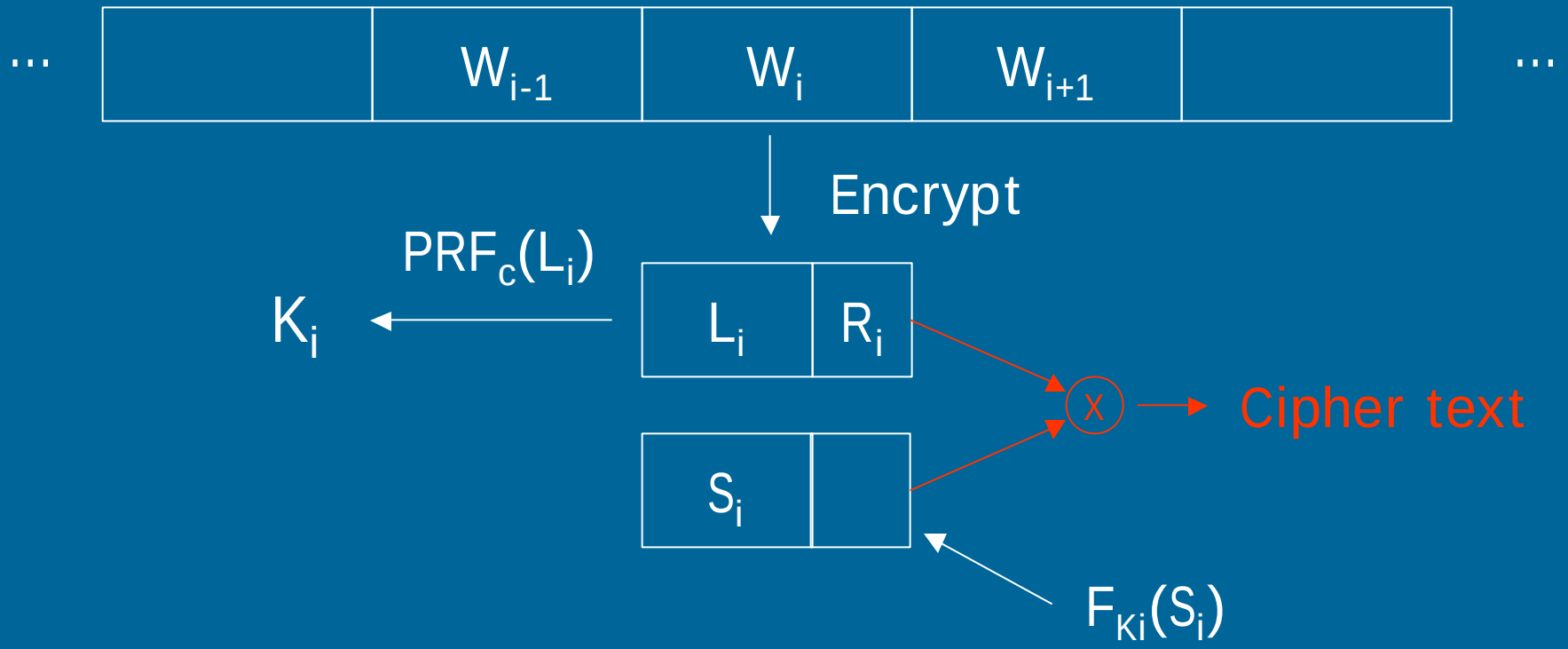
To Encrypt



4) Use a PRF with K_i to pad S_i

SSKE - Encrypt

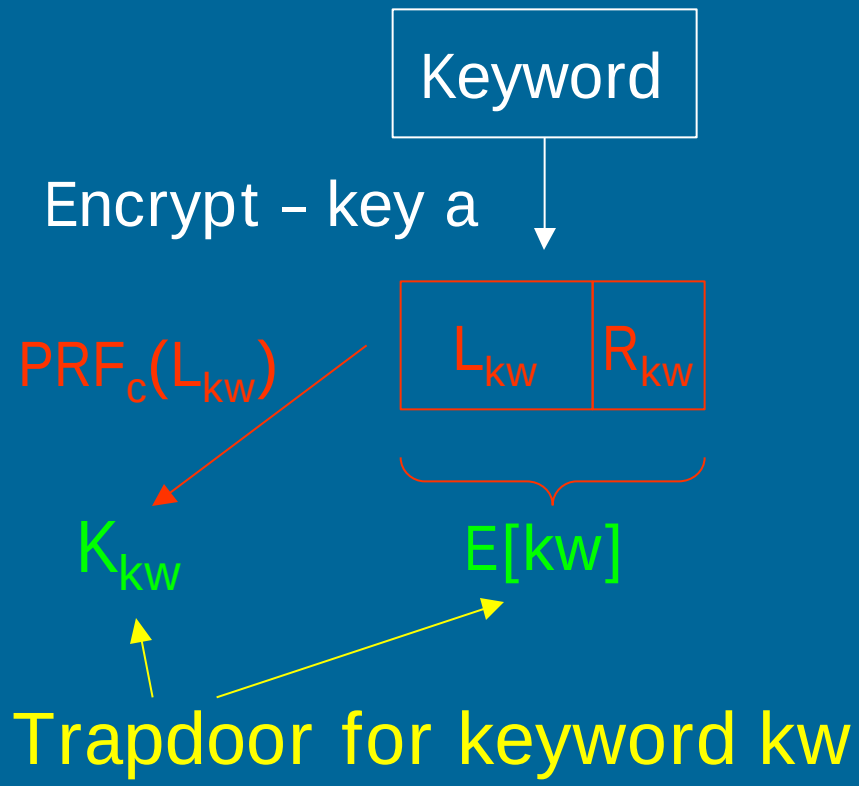
To Encrypt



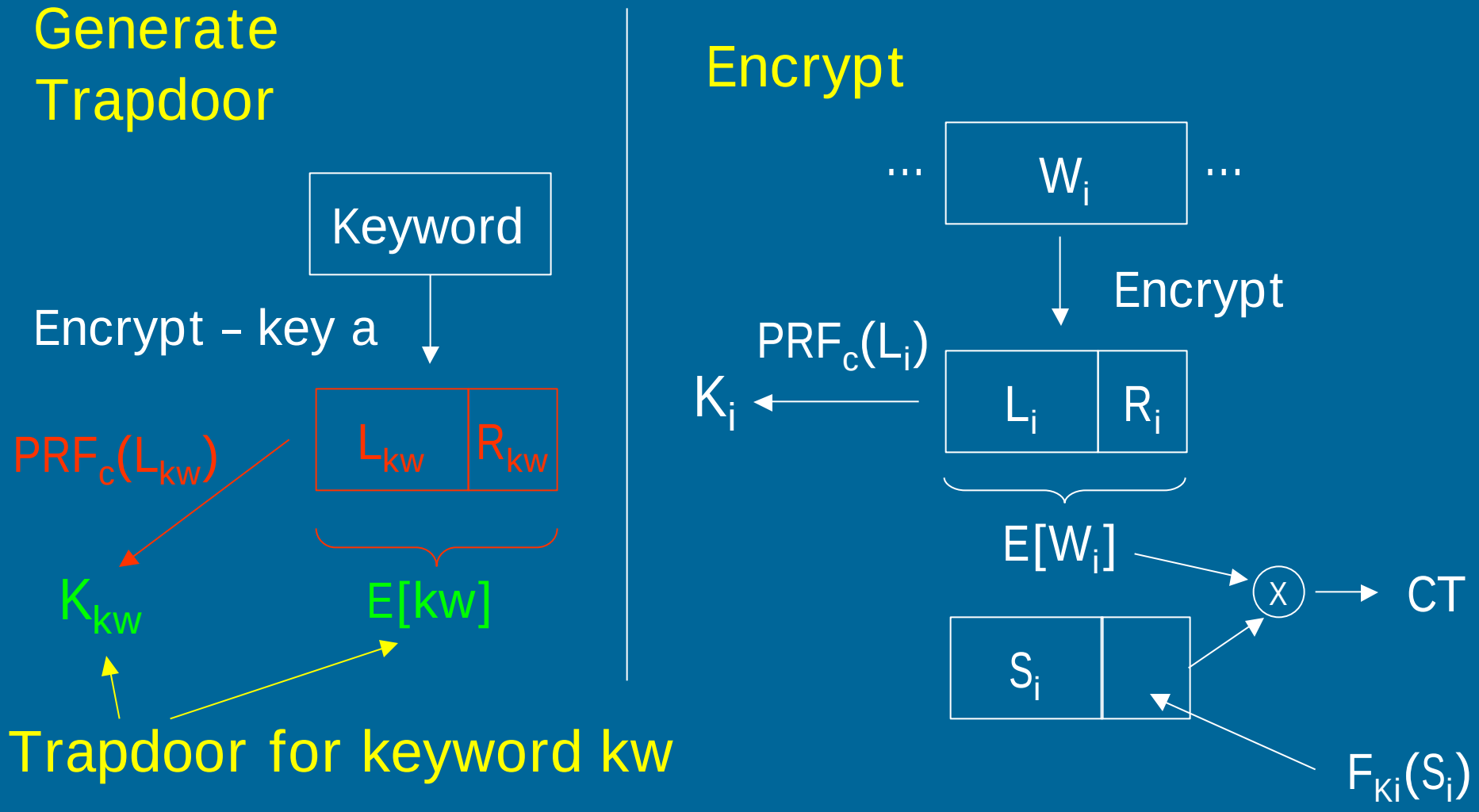
5) XOR two halves to form cipher text

SSKE - Trapdoor

Generate Trapdoor



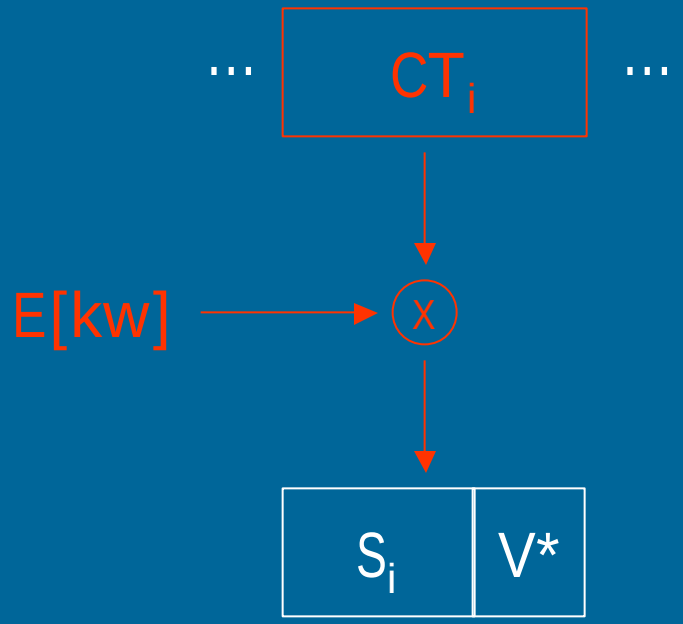
SSKE - Trapdoor



SSKE - Test

Server gets E[kw] and K_{kw} and scans document

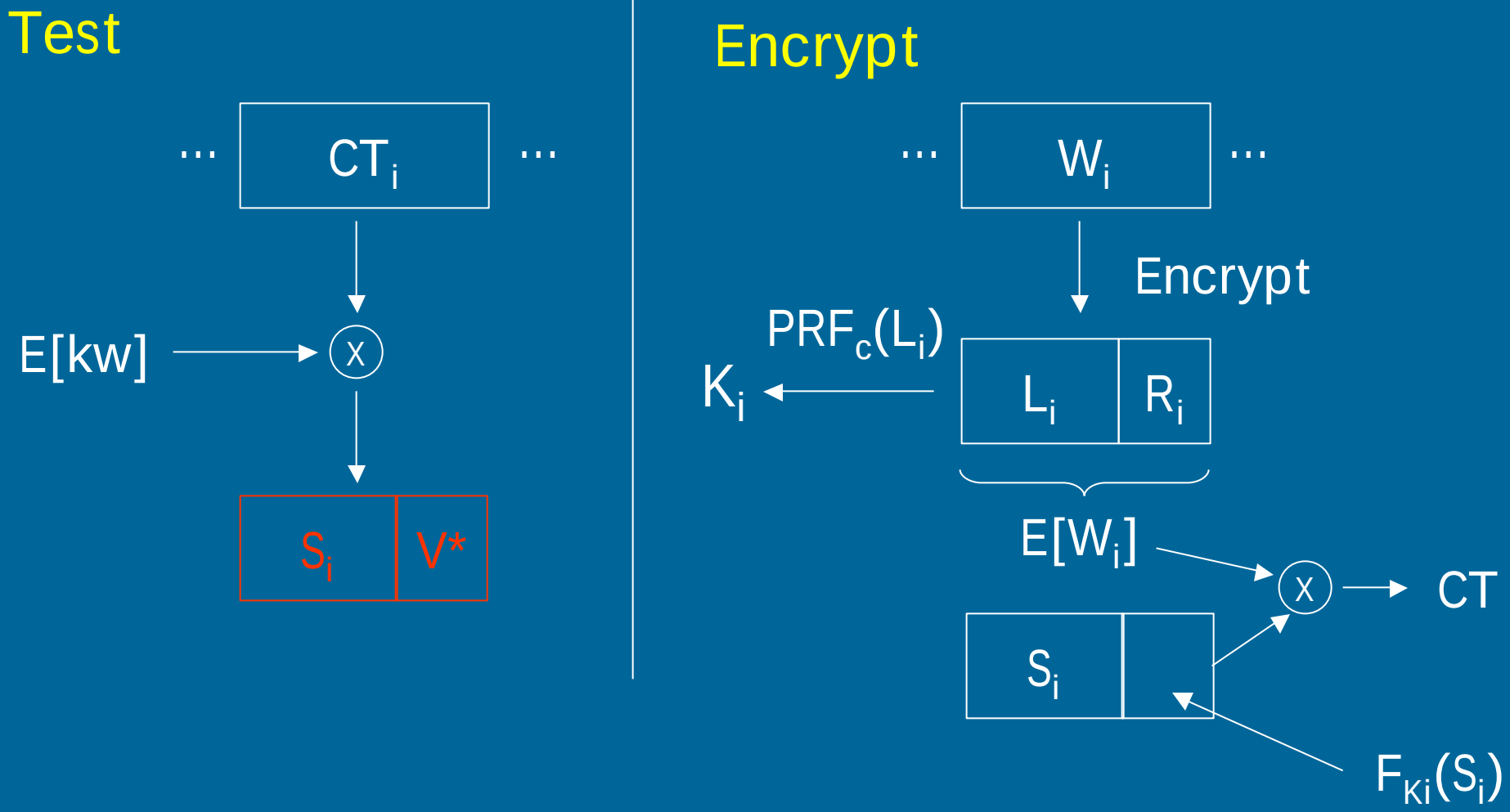
Test



1) XOR CT block with E[kw]

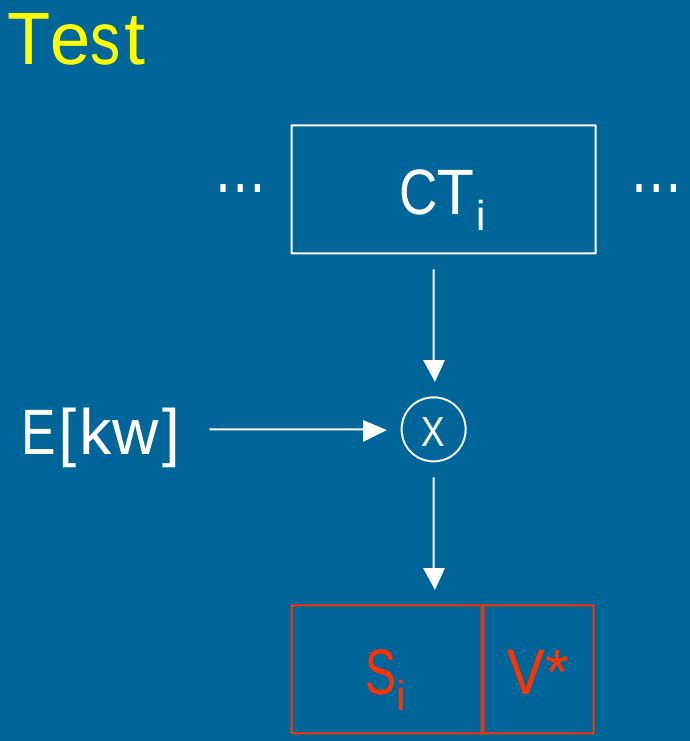
SSKE - Test

Server gets E[kw] and K_{kw} and scans document

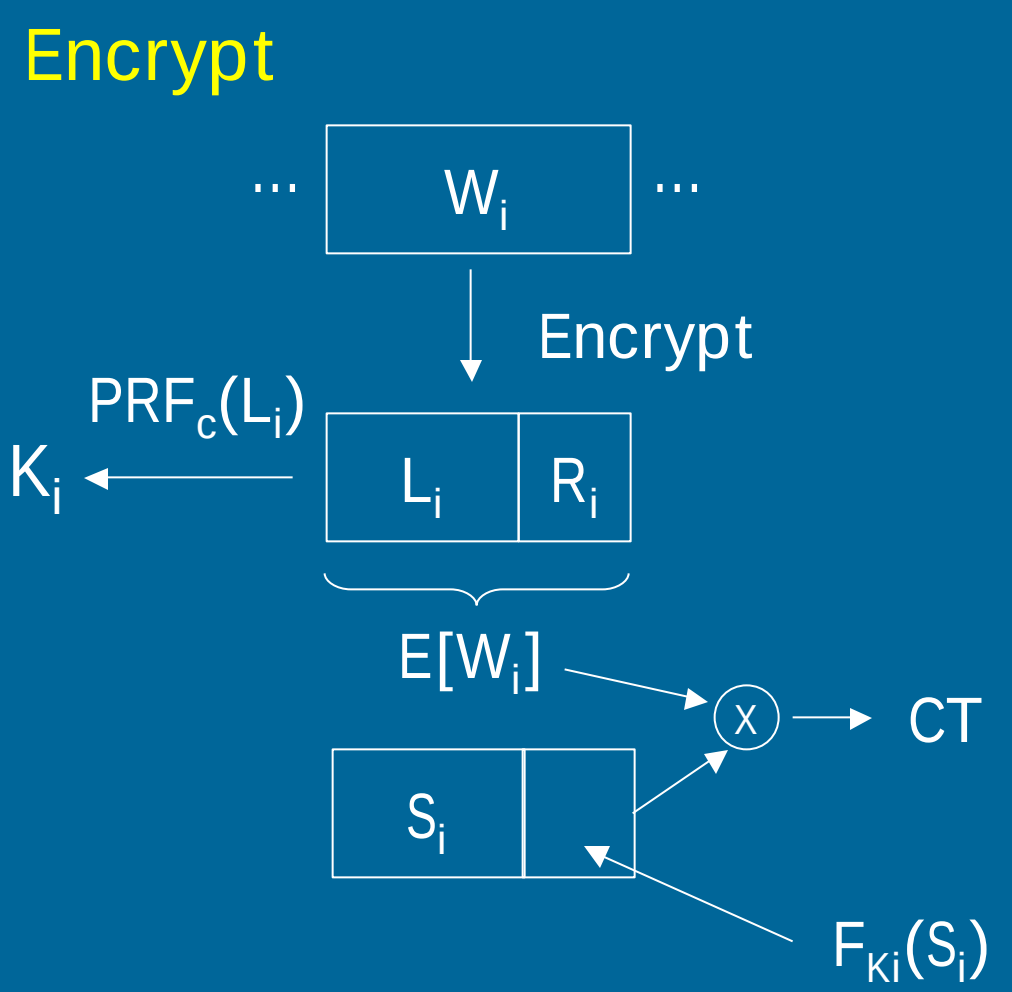


SSKE - Test

Server gets E[kw] and K_{kw} and scans document

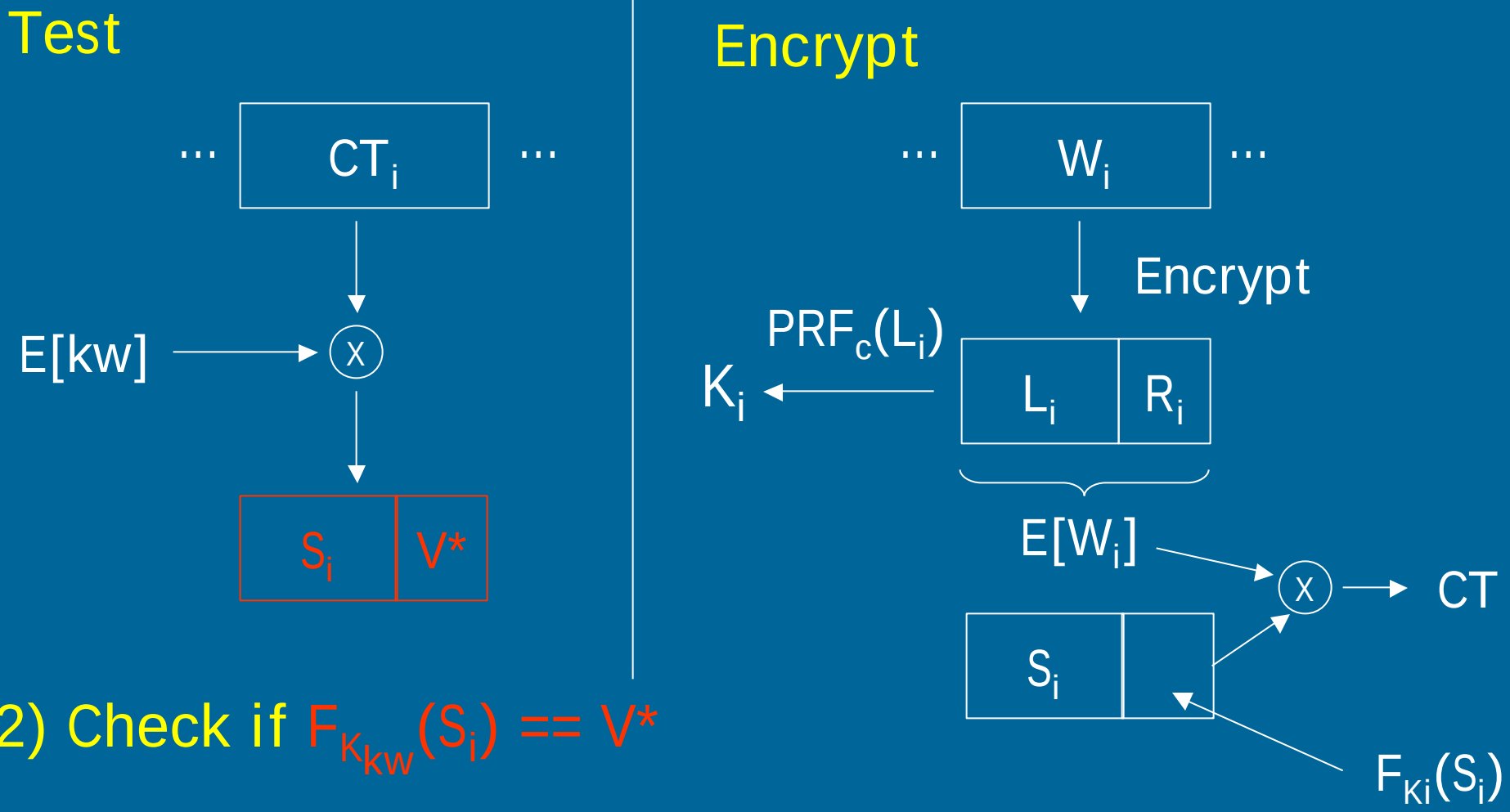


If CT_i = SSKE[kw], then $V^* = F_{K_{kw}}(S_i)$



SSKE - Test

Server gets E[kw] and K_{kw} and scans document



2) Check if $F_{K_{kw}}(S_i) == V^*$

Security of SSKE

- Proved that SSKE is a PRG
- Security related to that of PRF and PRG used in construction

Disadvantages

1. Work is linear in document size
2. Inelegant modifications to handle variable length words

SWP Keyword Index

Overview

- Hash table keyed by words
- Buckets — ptrs to docs

SWP Keyword Index

Overview

- Hash table keyed by words
- Buckets — ptrs to docs

Insecure updates

- Add new doc, update doc
  - bucket length changes in hash table
- Leaks info about doc word set

How to search on encrypted data?

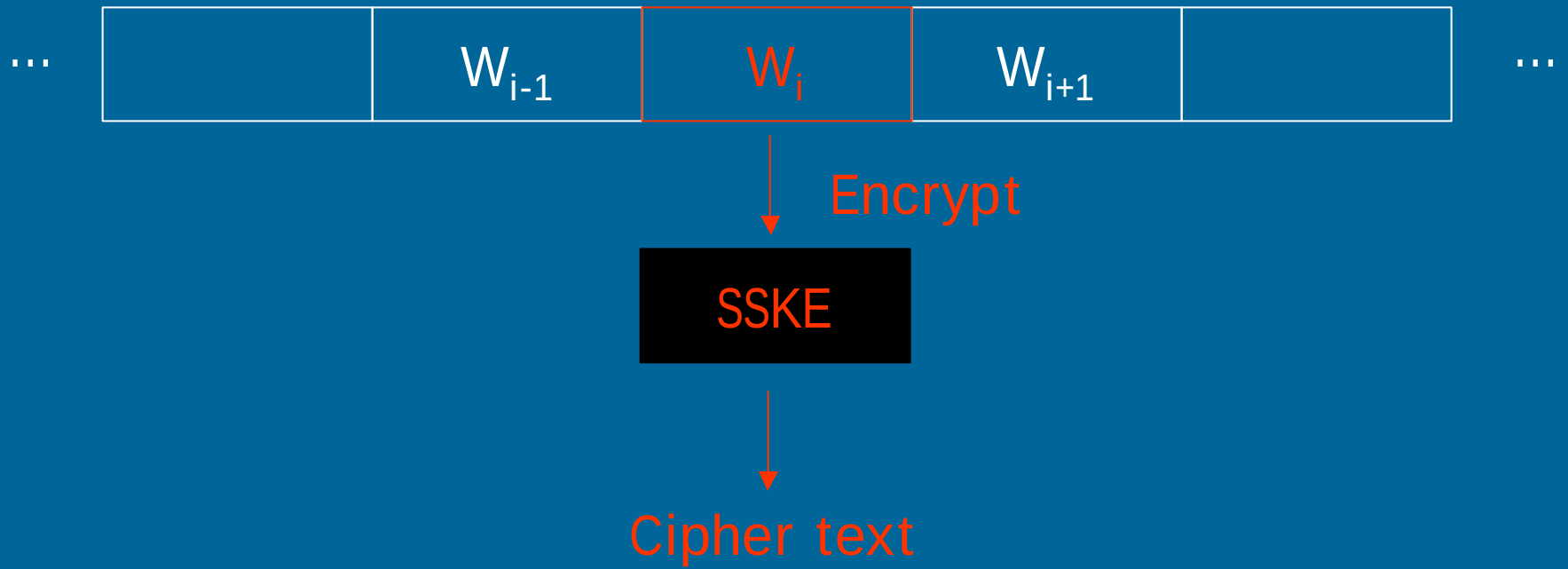
Solution 2 -

Searchable Public Key Encryption (SPKE)

D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano.

Linear Search with SSKE

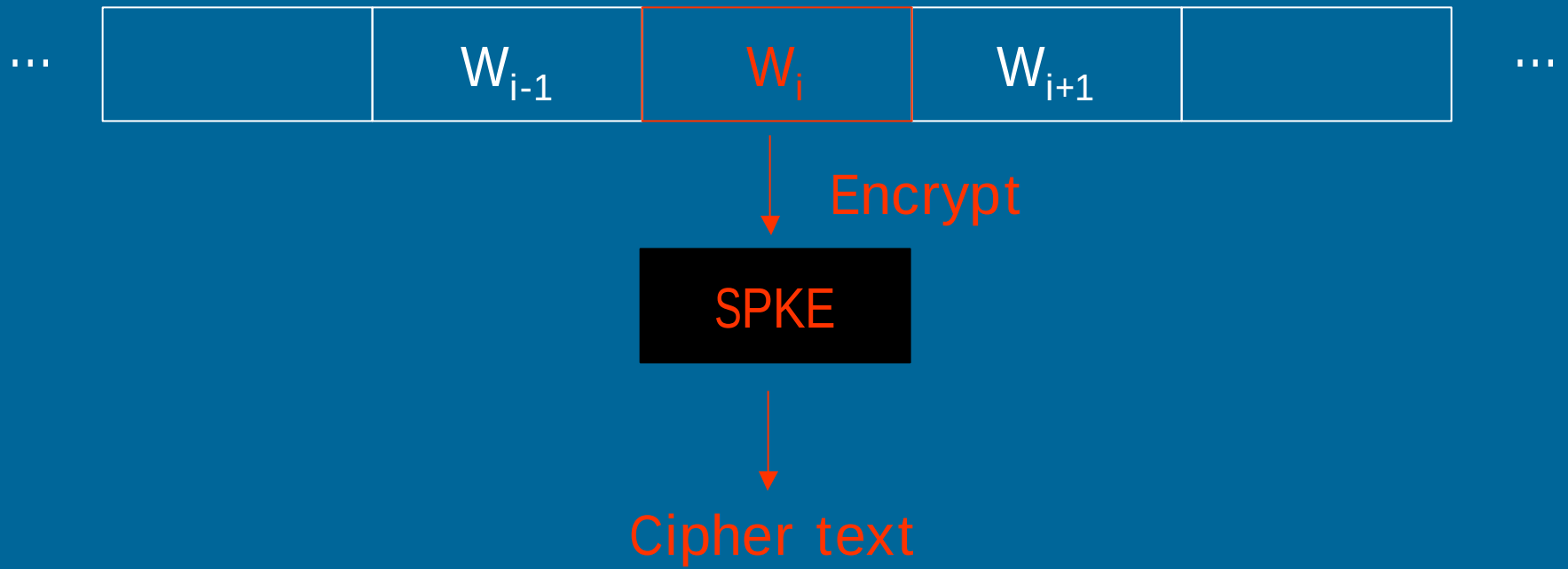
To Encrypt



Searchable Symmetric Key Encryption (SSKE)

Linear Search with SPKE

To Encrypt



Searchable Public Key Encryption (SPKE)

Overview

Motivation —

allow mail gateway to prioritize encrypted mail by keywords

3 constructions

1. Bilinear maps
2. Jacobi symbols
3. General trapdoor permutations

Bilinear Maps

G_1, G_2 - 2 groups of prime order p

Bilinear map –

- $e: G_1 \times G_1 \rightarrow G_2$
- For any $x,y \in [1,p]$, $e(g^x,g^y)=e(g,g)^{xy}$
- If g is a generator of G_1, then $e(g,g)$ is a generator of G_2.

SPKE Operations

1. Key Generation
2. Encrypt
3. Generate Trapdoor
4. Test for keyword

SPKE

Let $H_1 : \{0,1\}^* \rightarrow G_1$

KeyGen

1. Pick rand. $a \leftarrow Z_p^*$ and gen. $g \leftarrow G_1$
2. $A_{pub} = \{g, g^a\}$, $A_{priv} = a$

SPKE

Let $H_1 : \{0,1\}^* \rightarrow G_1$

KeyGen

1. Pick rand. $a \leftarrow Z_p^*$ and gen. $g \leftarrow G_1$
2. $A_{pub} = \{g, g^a\}$, $A_{priv} = a$

Encrypt - given A_{pub} and w

1. Pick rand. $r \leftarrow Z_p^*$
2. Compute g^r, g^{ar}, $t = e(H_1(w), g^{ar})$
3. CT — (g^r, t)

SPKE

Let $H_1 : \{0,1\}^* \rightarrow G_1$

KeyGen

1. Pick rand. $a \leftarrow Z_p^*$ and gen. $g \leftarrow G_1$
2. $A_{pub} = \{g, g^a\}$, $A_{priv} = a$

Encrypt - given A_{pub} and w

1. Pick rand. $r \leftarrow Z_p^*$
2. Compute g^r, g^{ar}, $t = e(H_1(w), g^{ar})$
3. CT — (g^r, t)

Trapdoor - given A_{priv} and w

1. Compute $T_w = H_1(w)^a$

SPKE

Let $H_1 : \{0,1\}^* \rightarrow G_1$

KeyGen

1. Pick rand. $a \leftarrow Z_p^*$ and gen. $g \leftarrow G_1$
2. $A_{pub} = \{g, g^a\}$, $A_{priv} = a$

Encrypt - given A_{pub} and w

1. Pick rand. $r \leftarrow Z_p^*$
2. Compute g^r, g^{ar}, $t = e(H_1(w), g^{ar})$
3. CT — (g^r, t)

Trapdoor - given A_{priv} and w

1. Compute $T_w = H_1(w)^a$

Test for w - given $T_w = H_1(w)^a$, CT=(A,B)

1. Check if $e(T_w, A) == B$

SPKE

Test for w - given $T_w = H_1(w)^a$, CT=(A,B)

1. Check if $e(T_w, A) == B$

SPKE

Test for w - given $T_w = H_1(w)^a$, CT=(A,B)

1. Check if $e(T_w, A) == B$
2. If (A,B) contains w,

   then $A = g^r$, $B = e(H_1(w), g^{ar})$

$\Rightarrow e(T_w, A) = e(H_1(w)^a, g^r)$

SPKE

Test for w - given $T_w = H_1(w)^a$, CT=(A,B)

1. Check if $e(T_w, A) == B$
2. If (A,B) contains w,

   then $A = g^r$, $B = e(H_1(w), g^{ar})$

$\Rightarrow e(T_w, A) = e(H_1(w)^a, g^r)$

$= e(H_1(w), g)^{ar}$

SPKE

Test for w - given $T_w = H_1(w)^a$, CT=(A,B)

1. Check if $e(T_w, A) == B$
2. If (A,B) contains w,

   then $A = g^r$, $B = e(H_1(w), g^{ar})$

$\Rightarrow e(T_w, A) = e(H_1(w)^a, g^r)$

$= e(H_1(w), g)^{ar}$

$= e(H_1(w), g^{ar})$

SPKE

Test for w - given $T_w = H_1(w)^a$, CT=(A,B)

1. Check if $e(T_w, A) == B$
2. If (A,B) contains w,

   then $A = g^r$, $B = e(H_1(w), g^{ar})$

$\Rightarrow e(T_w, A) = e(H_1(w)^a, g^r)$

$= e(H_1(w), g)^{ar}$

$= e(H_1(w), g^{ar})$

$= B$

Security

- Based on Bilinear Diffie-Hellman assumption
  - Given g, g^a, g^b, $g^c \in G_1$, hard to compute $e(g,g)^{abc}$
- Model — semantic security against chosen keyword attack (SS-CKA)

How to search on encrypted data?

Solution 3 -

Secure Indexes for Searching Efficiently on Encrypted Compressed Data

E.-J. Goh

Overview

Build secure index for documents

- Indexes give $O(1)$ search time

Build using

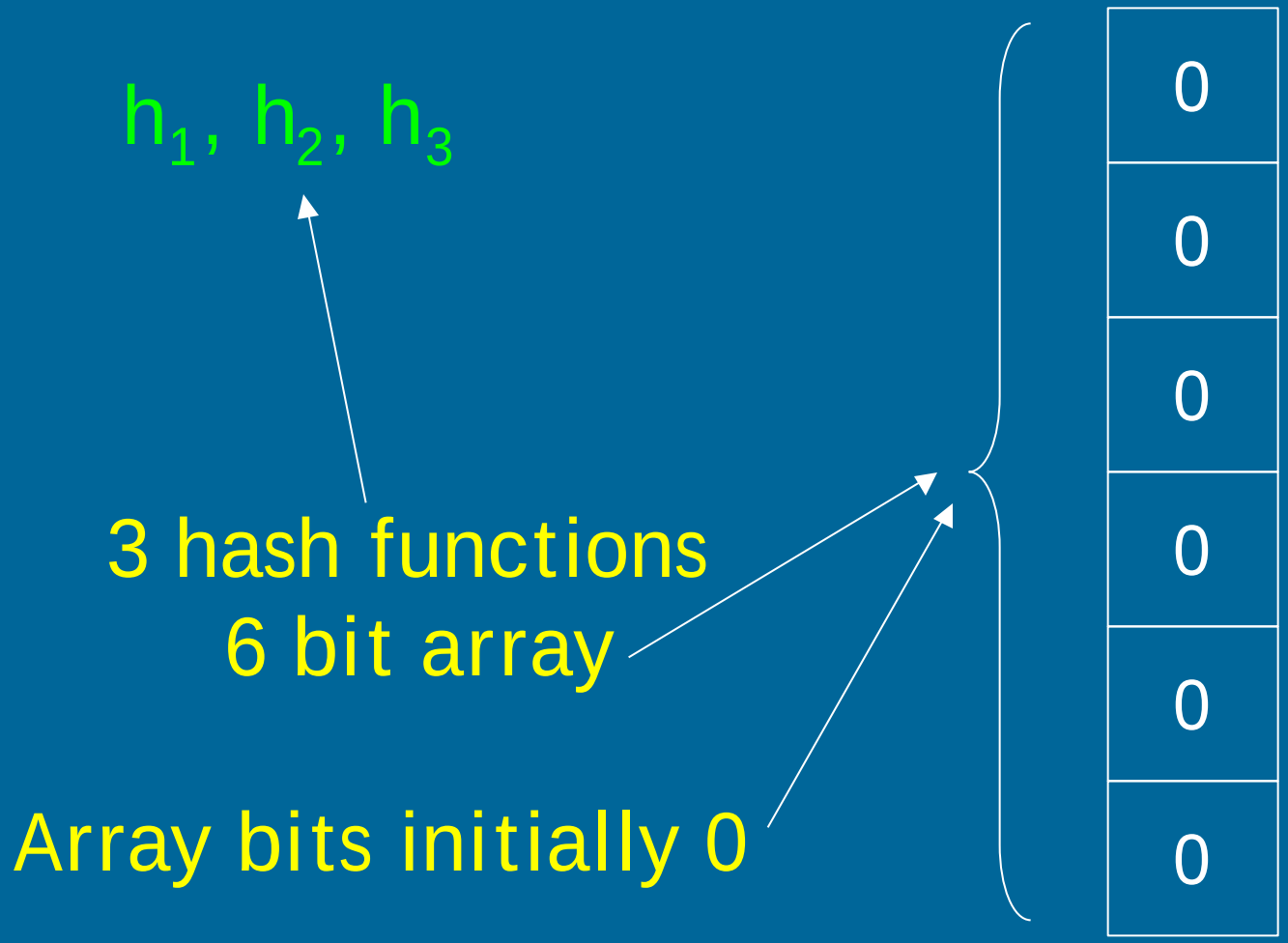
1. Bloom filters – efficient test for set membership
2. PRF – emulate "random functions"

Bloom Filters

A Bloom filter

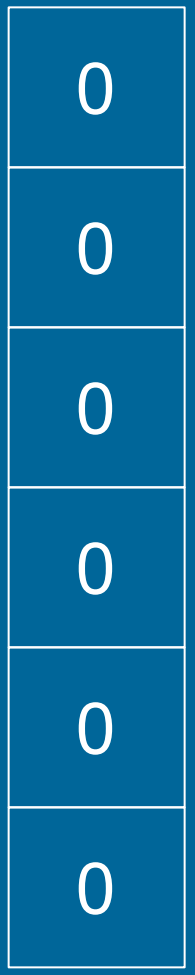
- Represents a set $S=\{s_1,\ldots,s_n\}$
- Is depicted by m bit array
- Uses r independent hash functions
  - $h_1,\ldots,h_r : \{0,1\}^* \rightarrow [1,m]$

Insertion Example



Insertion Example

To insert word x



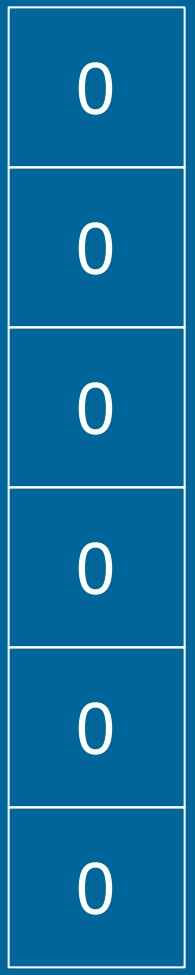
Insertion Example

To insert word x

$h_1(x) = 0$

$h_2(x) = 3$

$h_3(x) = 5$



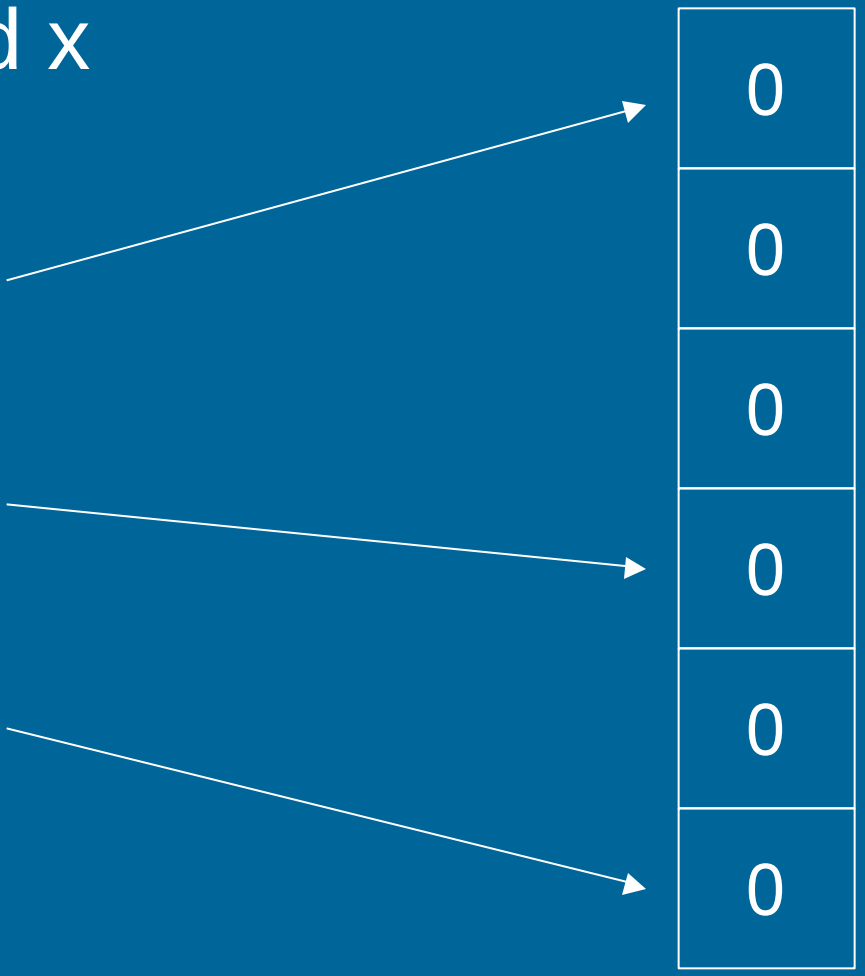
Insertion Example

To insert word x

$h_1(x) = 0$

$h_2(x) = 3$

$h_3(x) = 5$



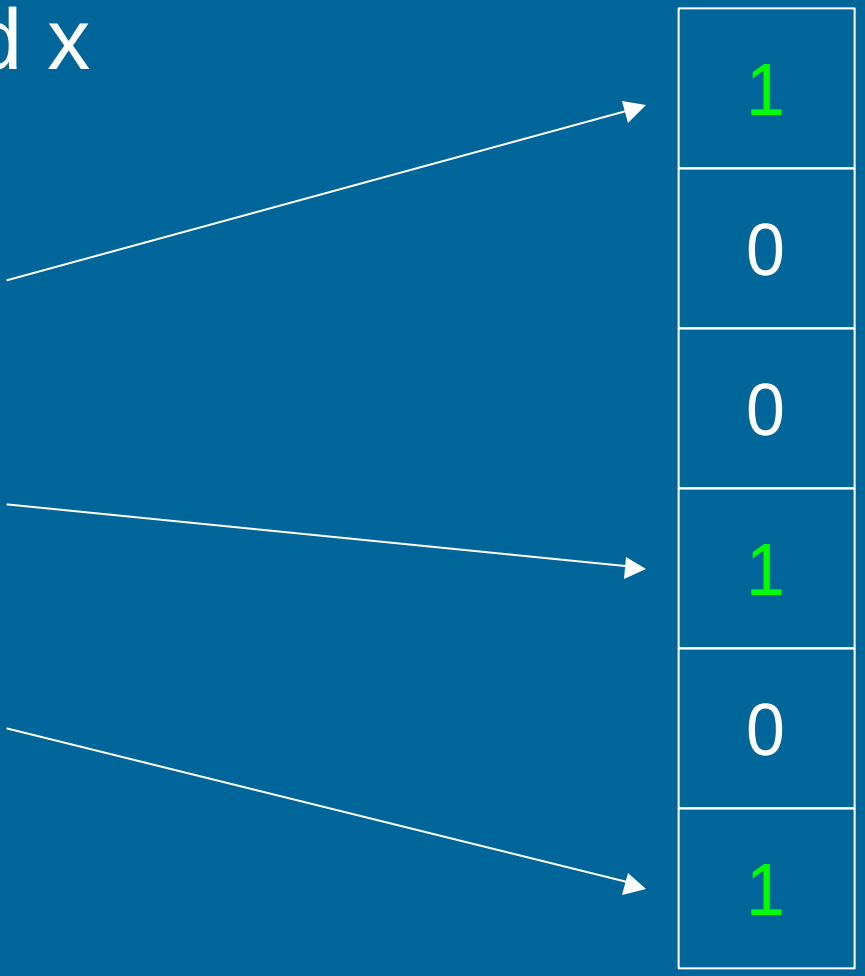
Insertion Example

To insert word x

$h_1(x) = 0$

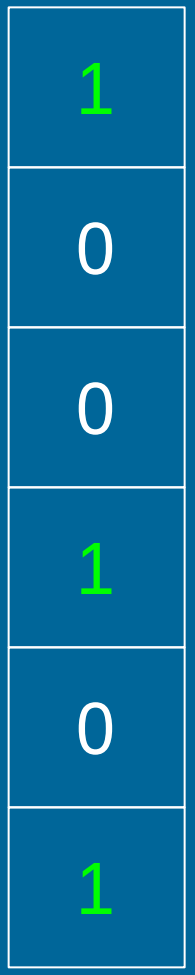
$h_2(x) = 3$

$h_3(x) = 5$



Insertion Example

To insert word y



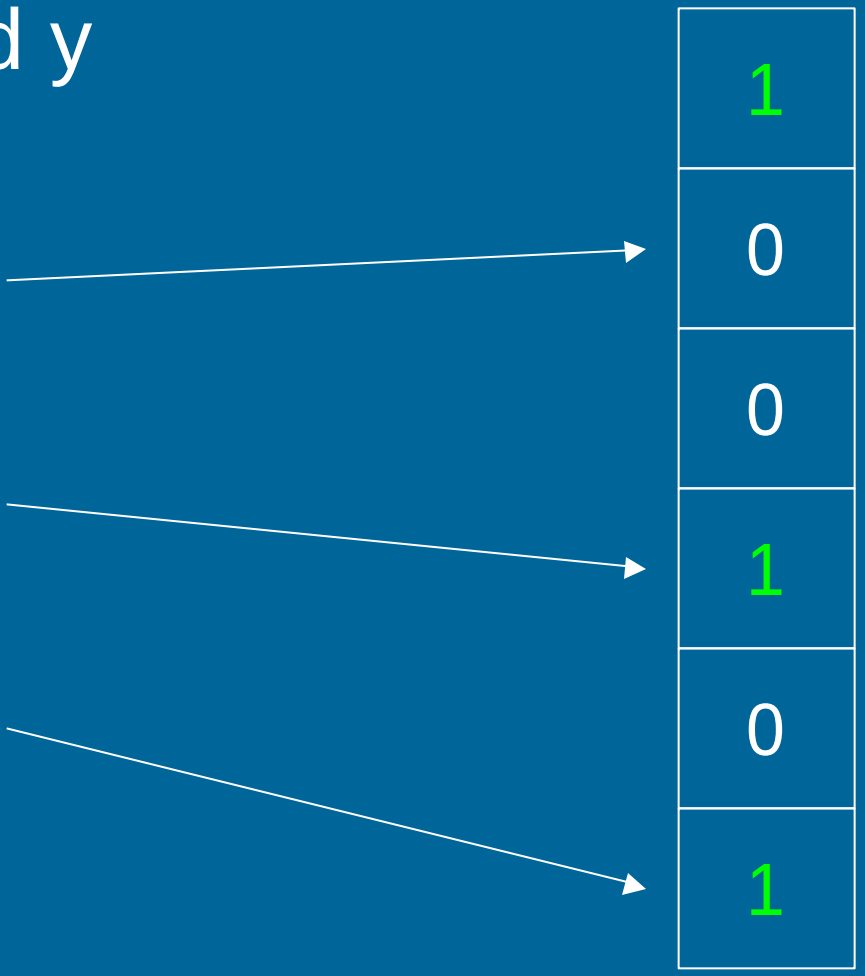
Insertion Example

To insert word y

$h_1(y) = 1$

$h_2(y) = 3$

$h_3(y) = 5$



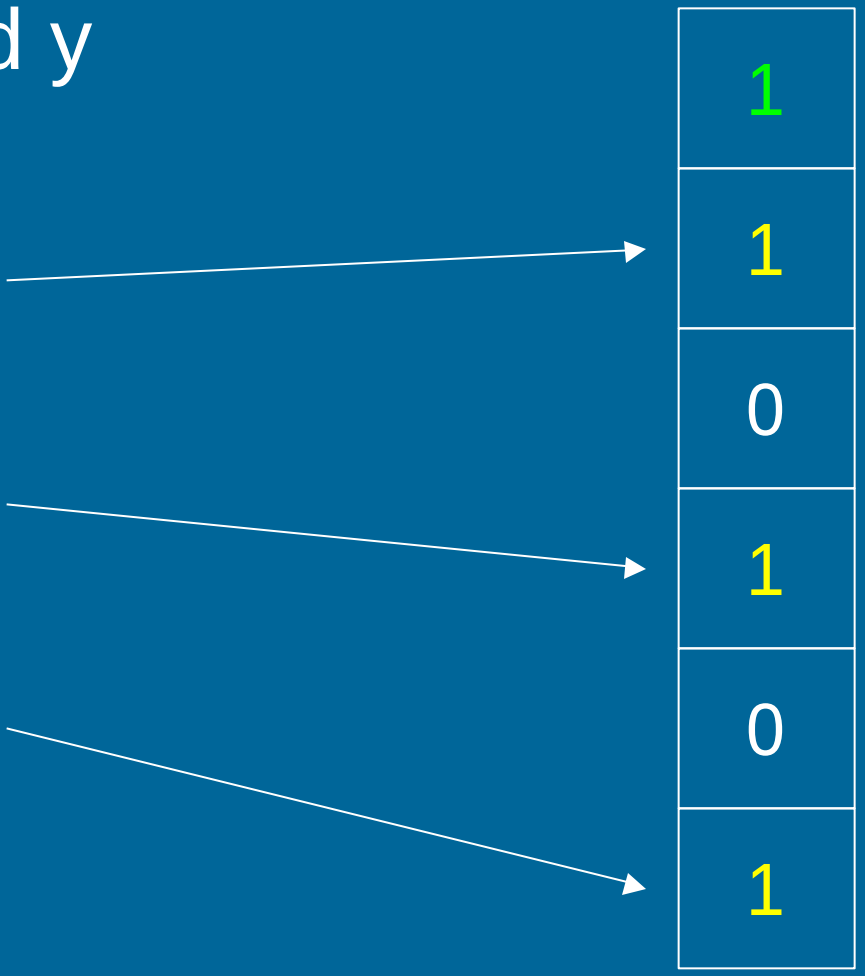
Insertion Example

To insert word y

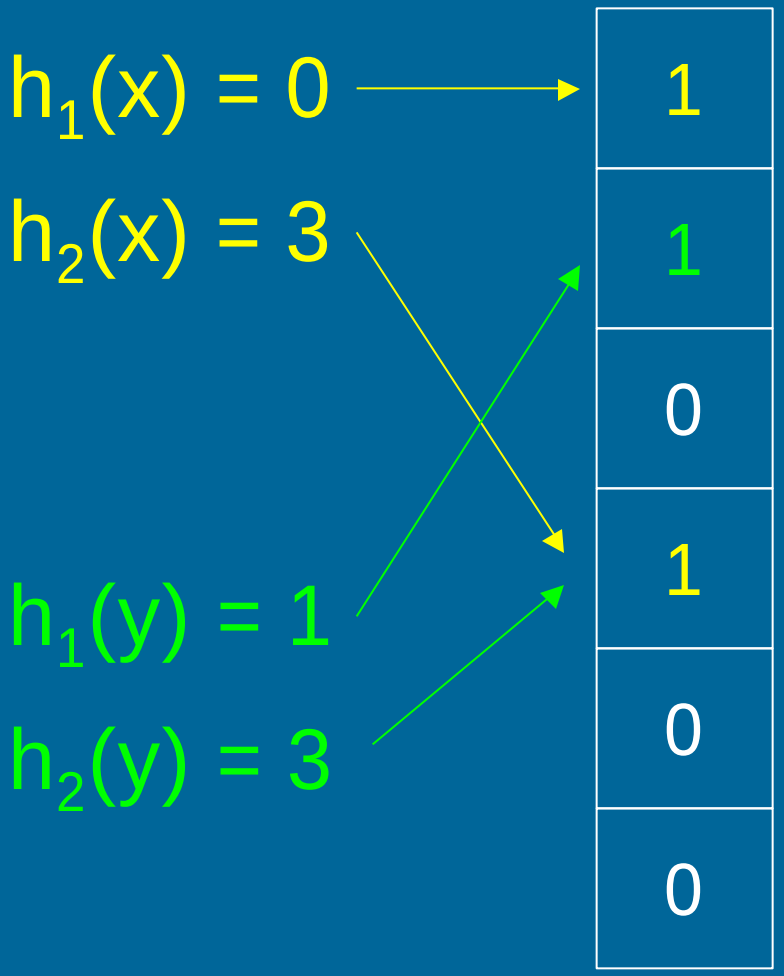
$h_1(y) = 1$

$h_2(y) = 3$

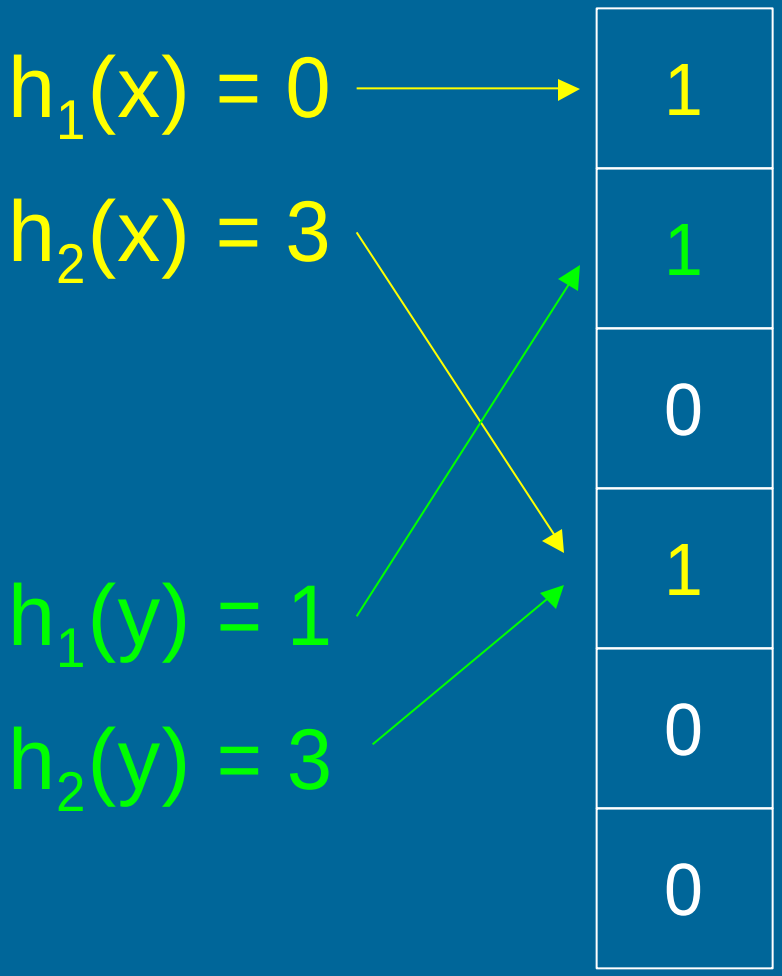
$h_3(y) = 5$



Testing Example

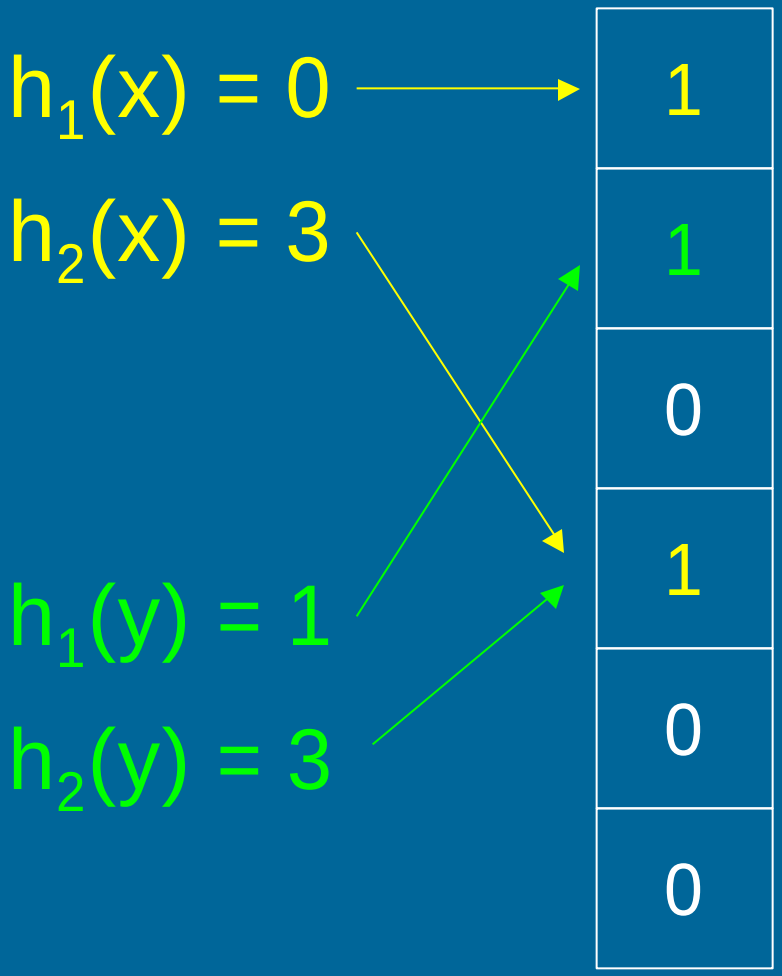


Testing Example



Does x belong?

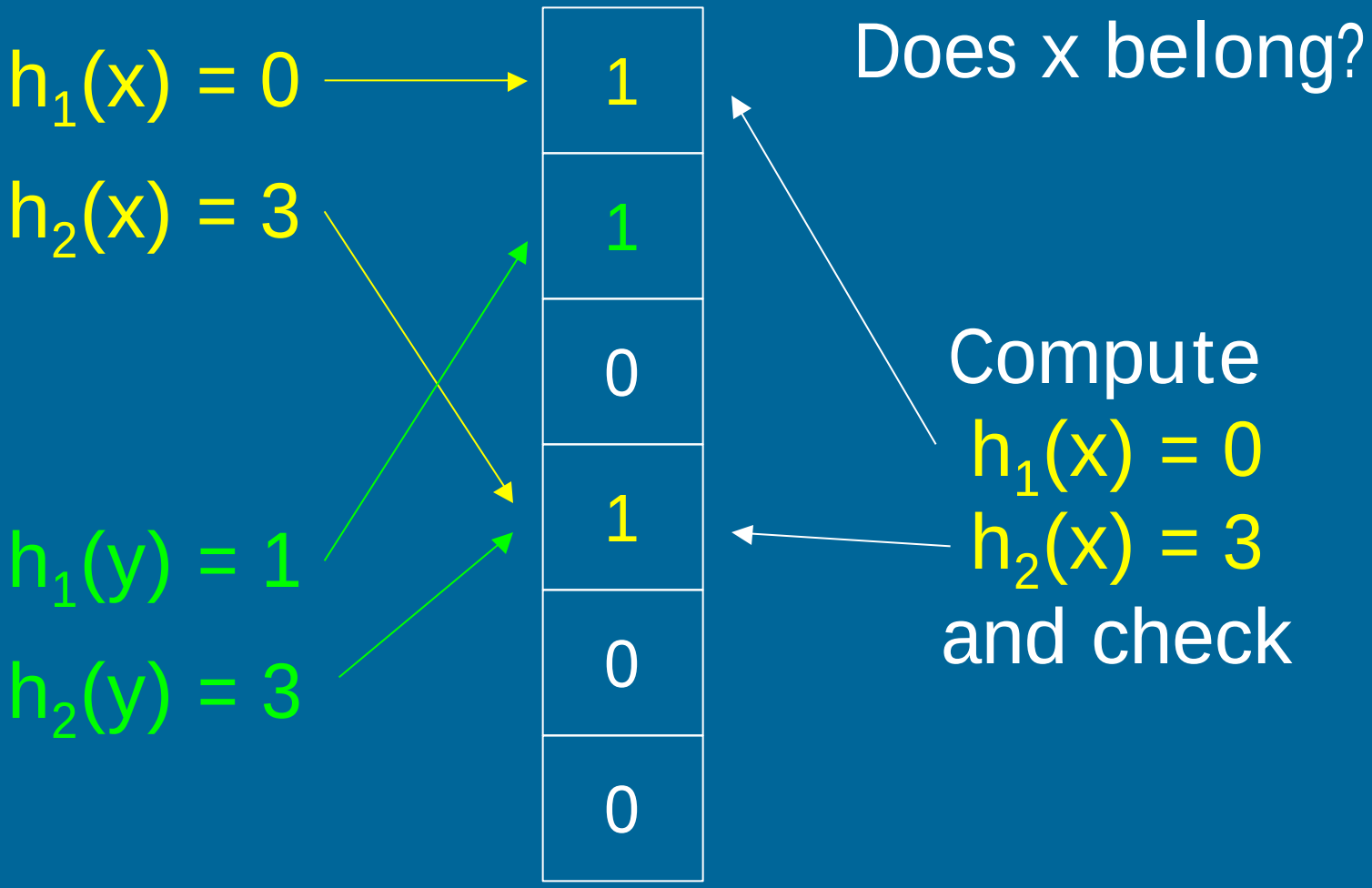
Testing Example



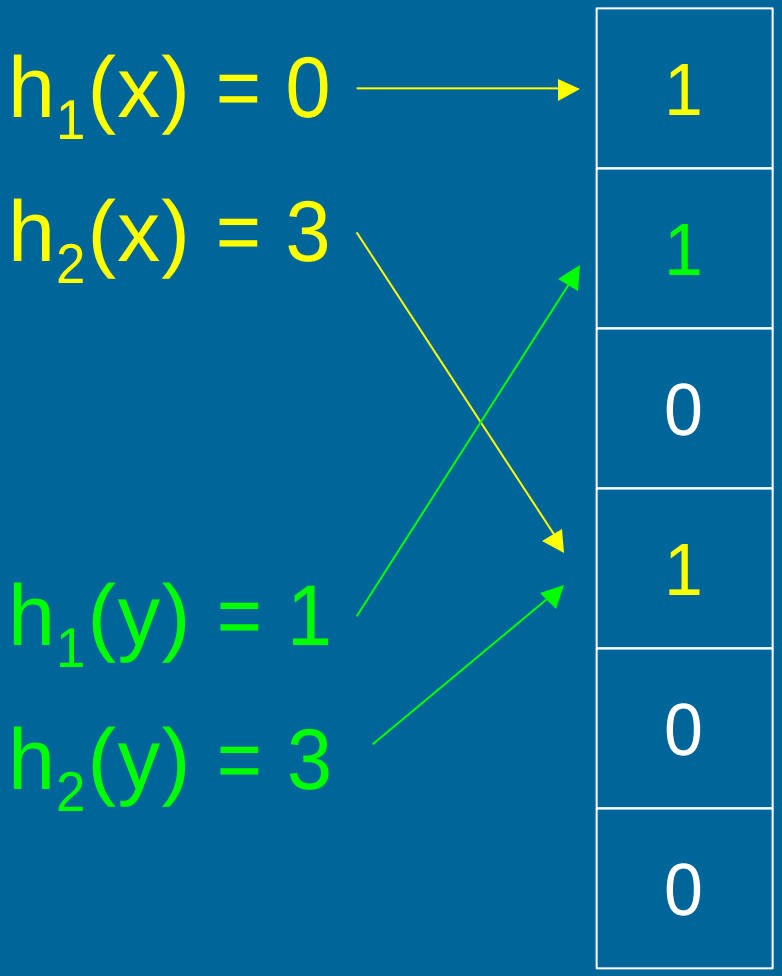
Does x belong?

Compute
$h_1(x) = 0$
$h_2(x) = 3$

Testing Example

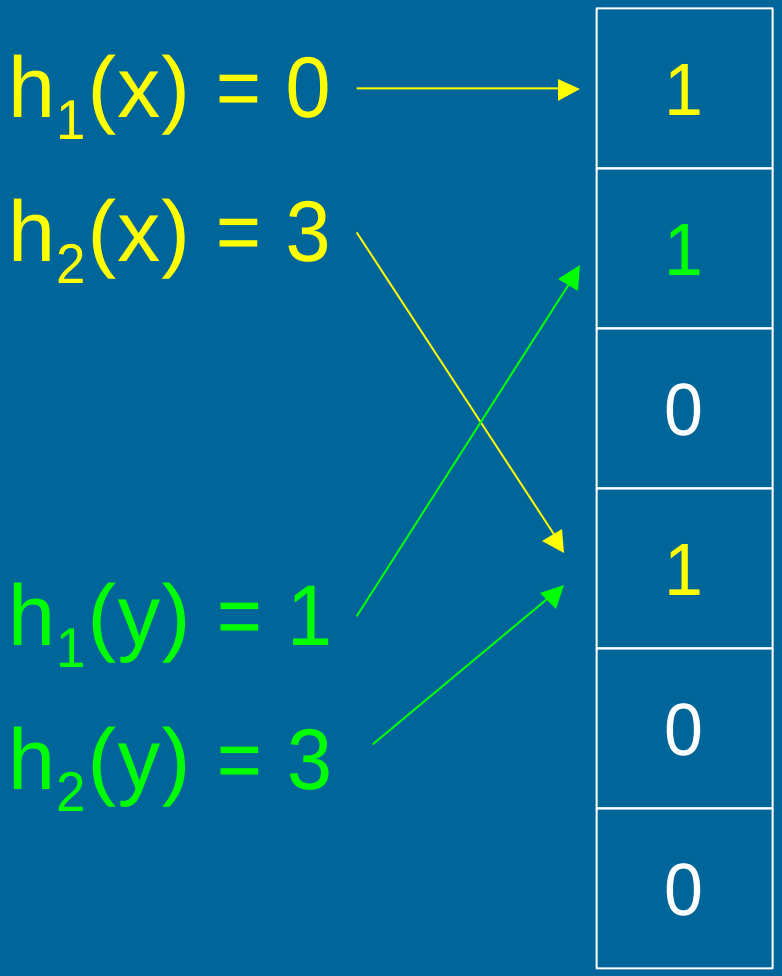


Testing Example



Does z belong?

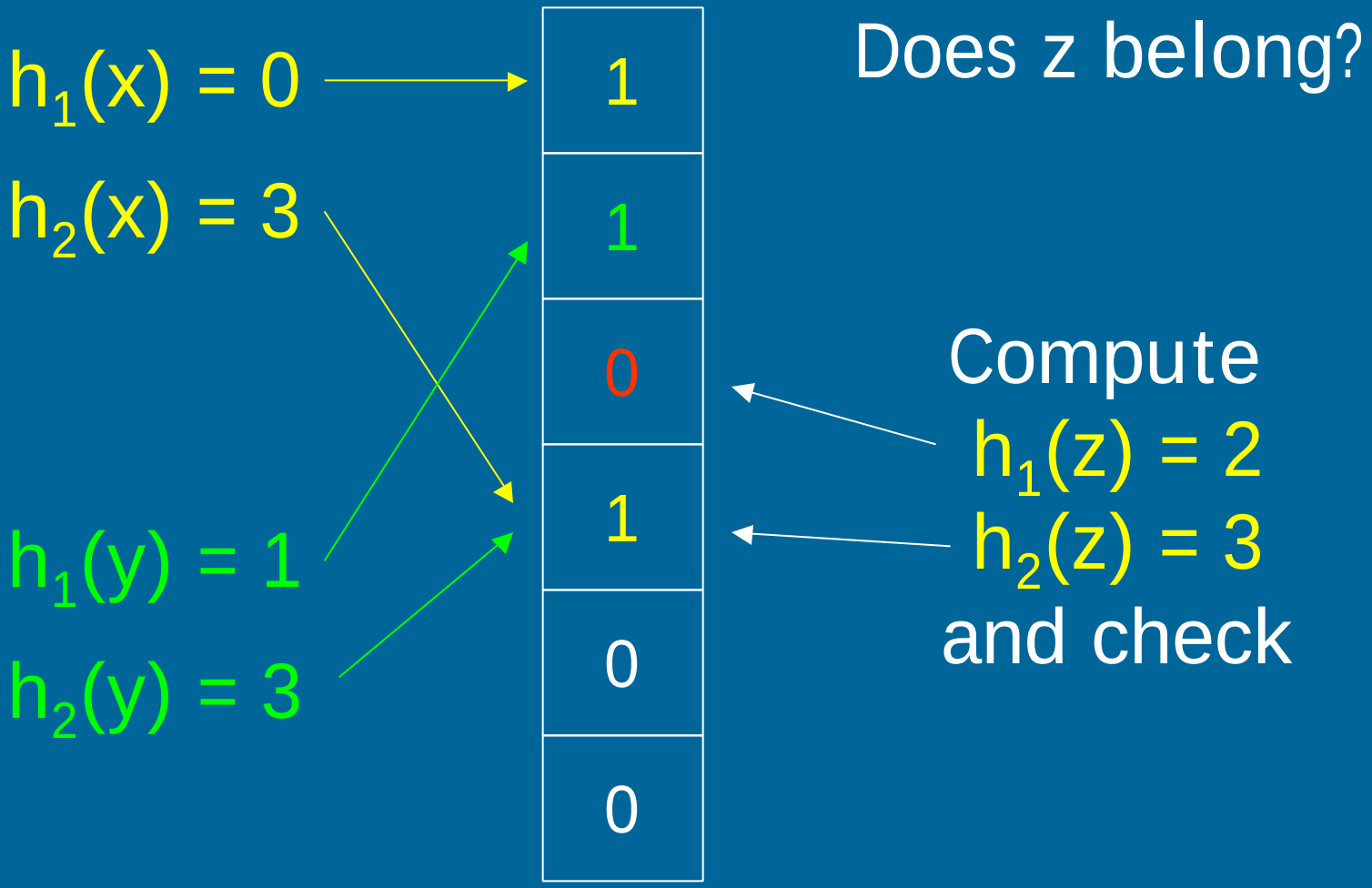
Testing Example



Does z belong?

Compute
$h_1(z) = 2$
$h_2(z) = 3$

Testing Example



Bloom Filter

False Positives

- If any tested array bit is 0, $a \notin S$
- Otherwise, a probably in S
- False positive rate depends on
  1. Number of hash functions
  2. Array size
  3. Number of elements in S

Pseudo-Random Functions

Intuitively

- PRF's indistinguishable from random functions
- Given $x_1,\ldots,x_m$ and $f_k(x_1),\ldots,f_k(x_m)$, adversary cannot predict $f_k(x_{m+1})$ for any x_{m+1}

Pseudo-Random Functions

A function $F : \{0,1\}^n \times \{0,1\}^s \rightarrow \{0,1\}^m$ is a (t,e,q)-prf if

- For *any* t time oracle algorithm A

$$\left| \Pr[A^F = 1] - \Pr[A^{RF} = 1] \right| < e$$

and A makes at most q queries.

Overview

- 1 Bloom filter (BF) per document
- Use PRF with r keys as r hash fxns

Notation

- Denote PRFs as $f_1, \ldots, f_r$
- Word Digest (WD)
  - $WD(x) = f_1(x), \ldots, f_r(x)$

Obvious Setup Algorithm

For a set of n documents

1. Generate suitable BF parameters
2. Scan each document —

   for each word x in document j

   insert WD(x) into document j's BF
3. Compress and encrypt docs
4. Transfer docs + indexes to server

Obvious Setup Algorithm

For a set of n documents

1. Generate suitable BF parameters
2. Scan each document —

   for each word x in document j

   insert WD(x) into document j's BF
3. Compress and encrypt docs
4. Transfer docs + index to server

Obvious Setup Algorithm

For a set of n documents

1. Generate suitable BF parameters
2. Scan each document —
   for each word x in document j
      insert WD(x) into document j's BF
3. Compress and encrypt docs
4. Transfer docs + index to server

Obvious Method is Insecure

Insecure to insert WD(x) in BF

- Compare doc BFs to determine doc similarity
- Learn doc contents from other doc indexes
- But analysis works only if same WD(x) across all docs

Obvious Method is Insecure

Insecure to insert WD(x) in BF

- Compare doc BFs to determine doc similarity
- Learn doc contents from other doc indexes
- But analysis works only if same WD(x) across all docs

Intuitive Solution

- Vary WD for each document
- But no key proliferation
  — use same r keys for all docs

Modified Setup

For a set of n documents

1. …
2. Assign and tag each document with an integer from [1,n]
3. Scan each document —

   for each word x in document j

   insert $f_{f_1(x)}(j),\ldots,f_{f_r(x)}(j)$ into doc j's BF
4. …

Modified Setup

For a set of n documents

1. …
2. Assign and tag each document with an integer from [1,n]
3. Scan each document —

   for each word x in document j

   insert $f_{f_1(x)}(j), \ldots, f_{f_r(x)}(j)$ into doc j's BF
4. …

Why More Secure?

Lemma

If f_k is a PRF, then $G(k) = f_k(1), \ldots, f_k(q)$ is a PRG

Why More Secure?

Lemma

If f_k is a PRF, then $G(k) = f_k(1), \ldots, f_k(q)$ is a PRG

Recall —

For each word x in document j

insert $f_{f_1(x)}(j), \ldots, f_{f_r(x)}(j)$ into doc j's BF

Why More Secure?

Lemma

If f_k is a PRF, then $G(k) = f_k(1), \ldots, f_k(q)$ is a PRG

Recall —

For each word x in document j

insert $f_{f_1(x)}(j), \ldots, f_{f_r(x)}(j)$ into doc j's BF

$\Rightarrow$ x's BF entry — PRG across all docs

$\Rightarrow$ no correlation!

Search

Search for keyword y

1. User — compute WD(y) = $f_1(y),\ldots,f_r(y)$
2. Send WD(y) to server

Search

Search for keyword y

1. User — compute WD(y) = $f_1(y),\ldots,f_r(y)$
2. Send WD(y) to server

1. Server — for each doc j,
   - Compute $f_{f_1(y)}(j),\ldots,f_{f_r(y)}(j)$
   - Test doc j's BF
2. Send user matching docs

Updates

1. Add document
   - Run setup alg. on new doc
2. Delete document
3. Update document
   - Assign new doc number
   - Regenerate BF using new number

$\wedge$ and $\vee$ Boolean Queries

$x \wedge y$ —

- check doc j BF for both $f_{f_1(x)}(j), \ldots, f_{f_r(x)}(j)$ and $f_{f_1(y)}(j), \ldots, f_{f_r(y)}(j)$

$x \vee y$ —

- check for either

$\Rightarrow$ Cost linear with size of query

Regular Exp. Queries

Limited set of regex queries

- Translate queries “ab[a-z]” to boolean queries “aba $\vee$ ... $\vee$ abz”

$\Rightarrow$ Cost linear with size of query

Other Nice Properties

1. Can handle compressed data
2. Indifferent to choice of cipher or compression algorithm
3. Variable Length Keywords
4. Simple Key Management

Heuristically Increasing Security

Recall — Search for keyword y

1. User — compute $WD(y) = f_1(y),\ldots,f_r(y)$
2. Send WD(y) to server

Heuristically Increasing Security

Recall — Search for keyword y

1. User — compute $WD(y) = f_1(y),\ldots,f_r(y)$
2. Send WD(y) to server

Instead of sending entire WD(y)

— send random *r/2* parts of WD(y)

Heuristically Increasing Security

Recall — Search for keyword y

1. User — compute WD(y) = $f_1(y),\ldots,f_r(y)$
2. Send WD(y) to server

Instead of sending entire WD(y)

— send random *r/2* parts of WD(y)

1. WD for repeated search of x looks diff
2. Heuristic — easy to detect in some cases

Occurrence Search

How to handle queries like “find all docs where foo occurs twice”?

Document setup

- prefix word with order of occurrence

Search query

- send digest for keyword “(2 || foo)”

Infrequent Keywords

Speed up search for infrequently occuring words

- Only if don't care about statistical analysis across documents
- build binary tree of doc BF

Security

Goal — Adversary learns no new info about a document from index

Security

Goal — Adversary learns no new info about a document from index

i.e. document P with n words

- m words known by adversary A
- n-m words unknown but A desires

Security

Goal — Adversary learns no new info about a document from index

i.e. document P with n words

- m words known by adversary A
- n-m words unknown but A desires

Achieved — A gains no info about n-m words from P's index even when have
  - plain text access to all docs + indexes
  - arbitrary queries to PRF on all words except n-m unknown words

Security Model

Semantic Security against Chosen Keyword Attack (SS-CKA)

— modeled as game between adversary and challenger

SS-CKA Game — Setup

Adversary A

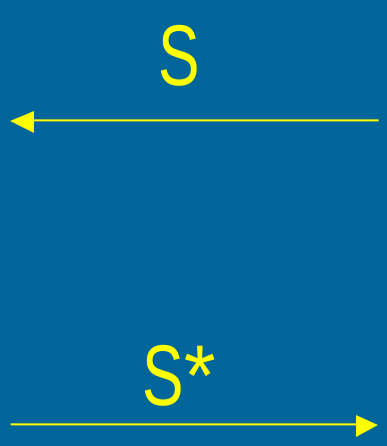
Challenger C

Create set S of q words

SS-CKA Game — Setup

Adversary A

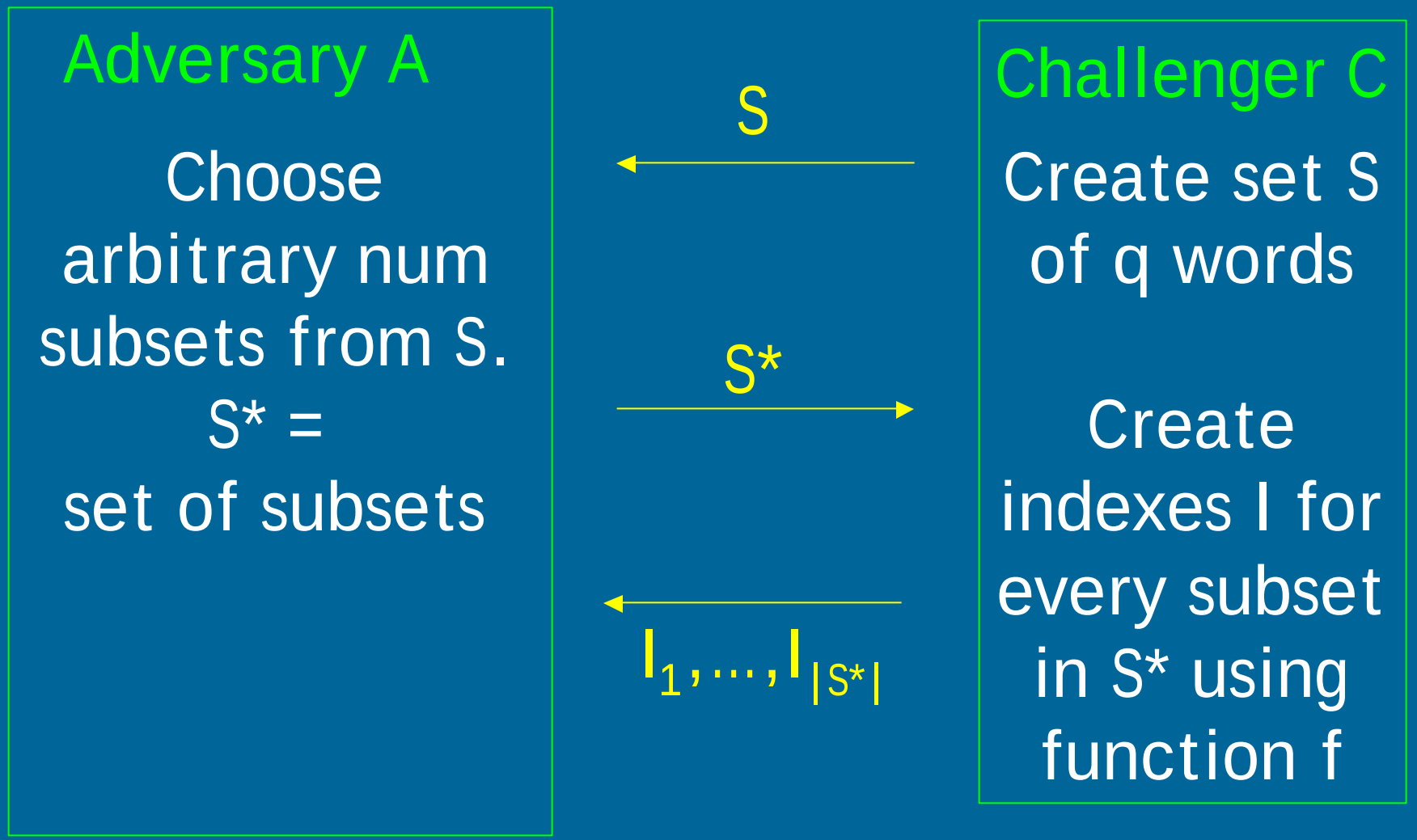
Choose arbitrary num subsets from S.
S* =
set of subsets



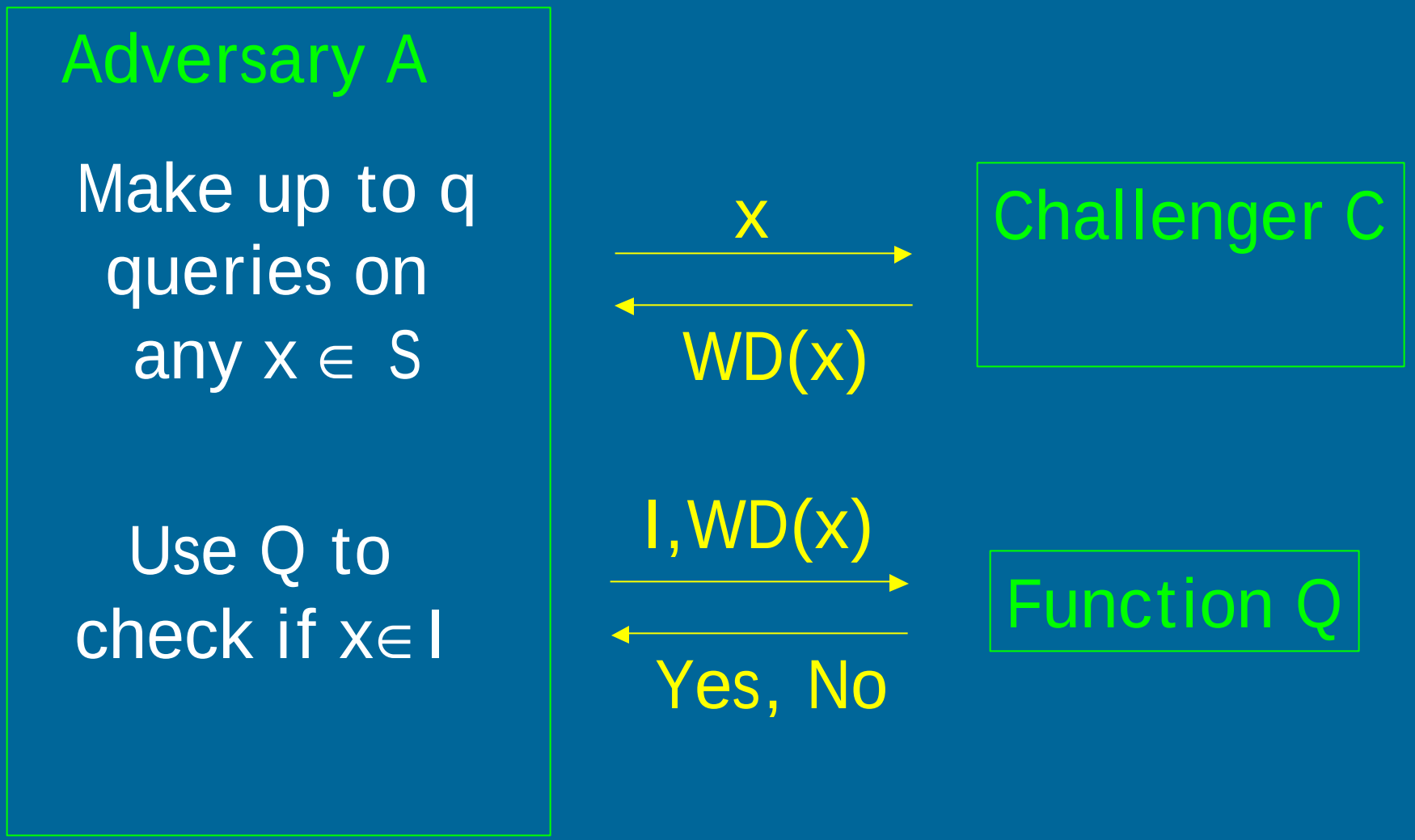
Challenger C

Create set S of q words

SS-CKA Game — Setup



SS-CKA Game — Queries



SS-CKA Game — Challenge

Adversary A

Choose $P \in S^*$

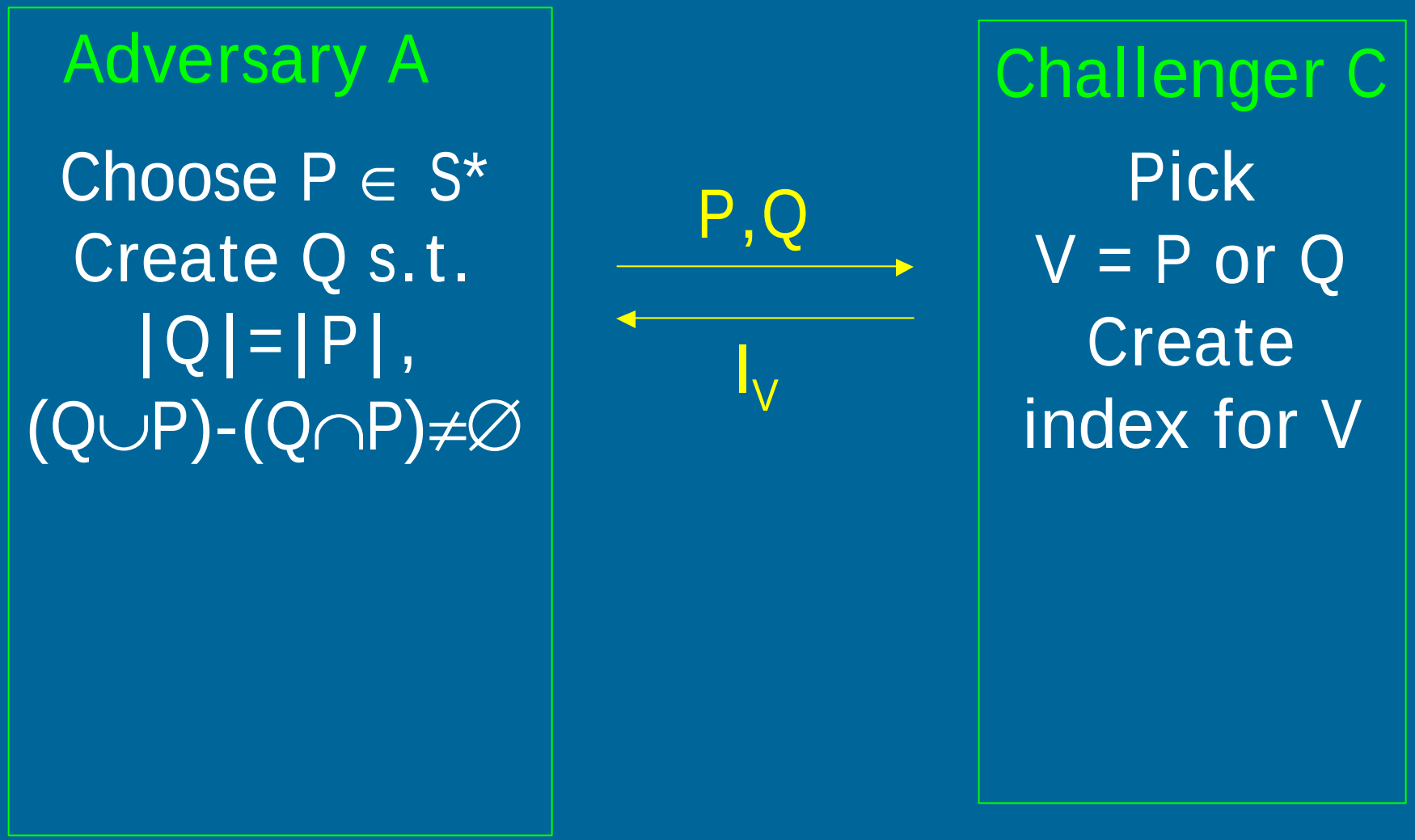
Create Q s.t.

$|Q|=|P|$,

$(Q \cup P)-(Q \cap P) \neq \emptyset$

Challenger C

SS-CKA Game — Challenge

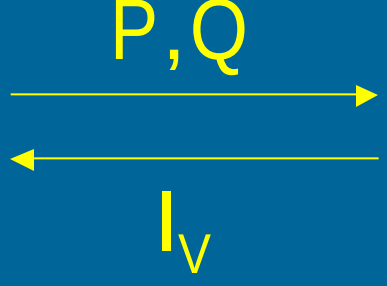


SS-CKA Game — Challenge

Adversary A

Choose $P \in S^*$
Create Q s.t.
$|Q|=|P|$,
$(Q \cup P)-(Q \cap P) \neq \emptyset$

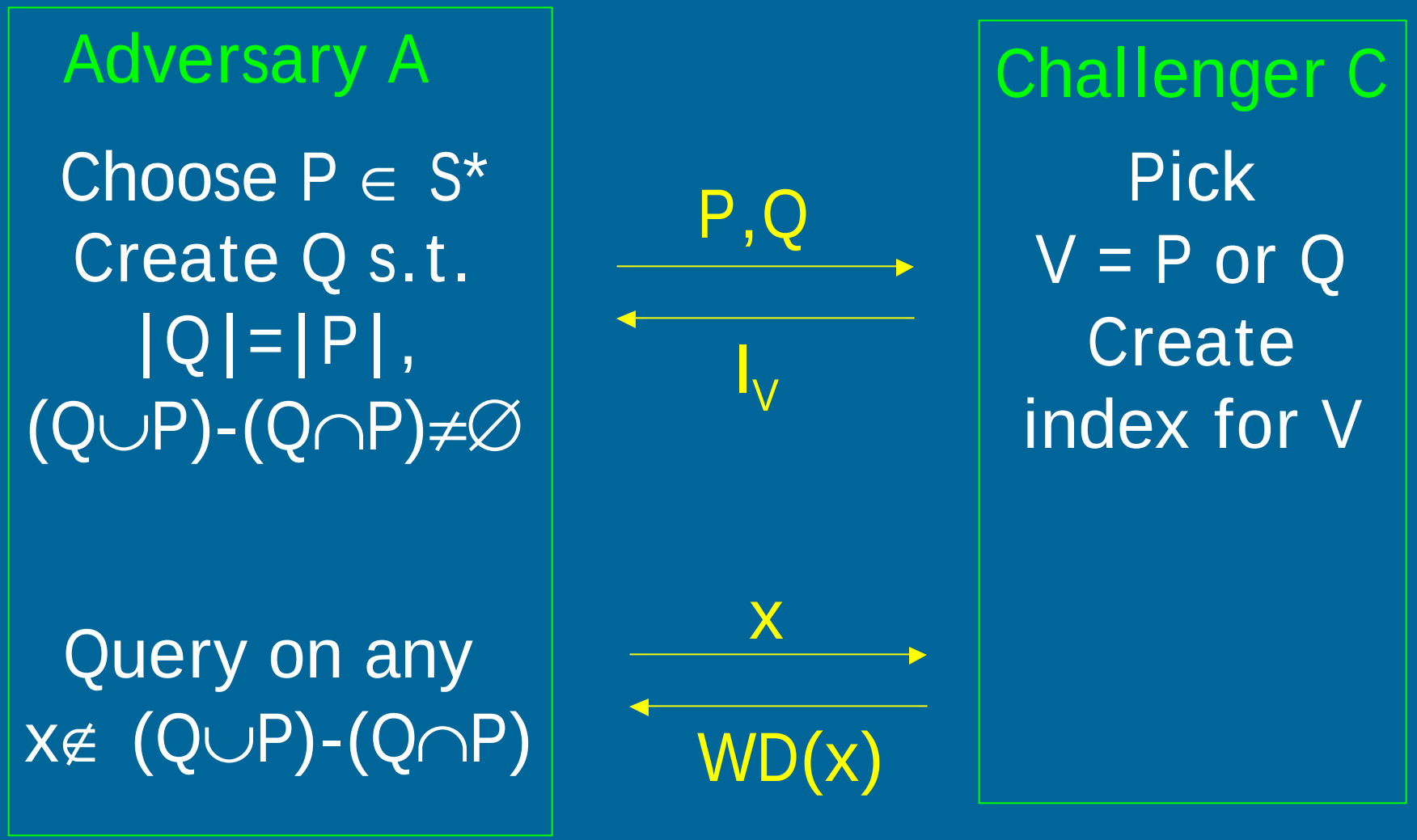
Challenge
Determine if
V = P or Q



Challenger C

Pick
V = P or Q
Create
index for V

SS-CKA Game — Challenge



SS-CKA Game

- A runs for *t* time and decides if V is P or Q
- A's advantage in winning is

$$e = \left| \Pr[\text{Guess} == V] - \tfrac{1}{2} \right|$$

Security Theorem

Theorem

If Bloom filter construction uses a (t, e, q)-PRF, then the Bloom filter is a (t, e, q)-SS-CKA index.

Bloom Filter Sizes

- Size of Bloom filter depends on num unique words in all docs
  - Calgary Corpus —
    - 437501 words, 22425 unique
    - For false positive 1 in 1024, use 10 hash fxns to give BF size 58 kB

Bloom Filter Sizes

Size of Bloom filter depends on num unique words in all docs

- Calgary Corpus —
  - 437501 words, 22425 unique
  - For false positive 1 in 1024, use 10 hash fxns to give BF size 58 kB
- IPSEC newsgroup —
  - 463869 words, 10014 unique
  - For false positive 1 in 1024, use 10 hash fxns to give BF size 29 kB

Sparse Array Encoding

- If large set of unique words over all docs but each doc small
  - Small docs $\Rightarrow$ small number of unique words per doc
  - Store using sparse array encoding

Sparse Array Encoding

- If large set of unique words over all docs but each doc small
  - Small docs $\Rightarrow$ small number of unique words per doc
  - Store using sparse array encoding
- e.g. 2.5 mil unique words but each doc contains only 5k unique words
  - BF — 4 MB
  - Store only 1's (instead of 0's) — BF 15 kB

Related Work

- Private Information Retrieval (PIR)
  - User — want cell j of bit vec. on server
  - PIR — get cell j without server knowing j
- Oblivious RAMs
  - Hide data access patterns