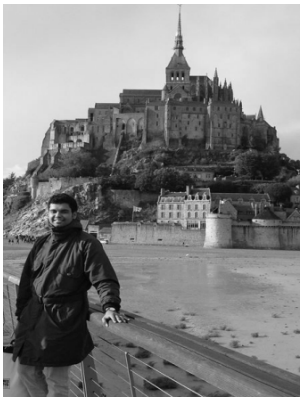
Can hardware make searchable encryption less hard?:

Result-Pattern Hiding Searchable Encryption for Conjunctive Queries

Professor
Debdeep Mukhopadhyay

Secured Embedded Architecture Laboratory (SEAL)
Department of Computer Science and Engineering
IIT Kharagpur
debdeep@cse.iitkgp.ernet.in

PHD STUDENTS INVOLVED…



Sikhar Patranabis

Pursuing PhD Candidate

Secured Embedded Architecture Laboratory (SEAL),

Department of Computer Science and Engineering, IIT Kharagpur.

Arnab Bag

Pursuing PhD Candidate

Secured Embedded Architecture Laboratory (SEAL),

Department of Computer Science and Engineering, IIT Kharagpur.

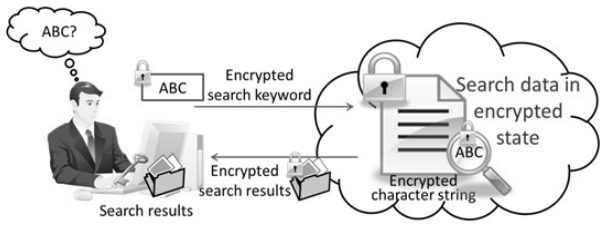
SECURED EMBEDDED ARCHITECTURE LABORATORY

VERTICAL DISCRETE ALGORITHM HORIZONTAL TIMING SECURITY COMPUTATION KHUDRA OPTIMIZATION MICKEY **ATTACK** MICRO GALOIS TVLA PARALLEL FIBONACCI PUF DEGREE ERROR TEMPLATE WATERMARKING BLOCK PRIME HASH GROUP PREDICTOR SBOX POWER ECC RSA COMMUNICATION FIELD SWARM ENTROPY SUPPORT SYSTEM LFSR CIPHER FAULT GRAIN CACHE MULTICORE COMPUTING THEORY CORRECTION BRANCH CRYPTANALYSIS FUNCTION STREAM XOR VECTOR **COUNTERMEASURE** BYTE DECRYPTION MACHINE LOGARITHM TROJAN PERMUTATION RING EMBEDDED AES CORRELATION SIDE-CHANNEL CODING ALGEBRAIC

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

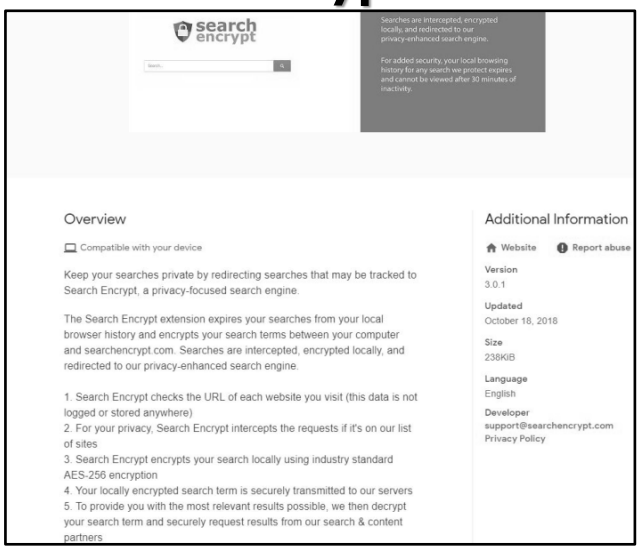
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

The Seal Family

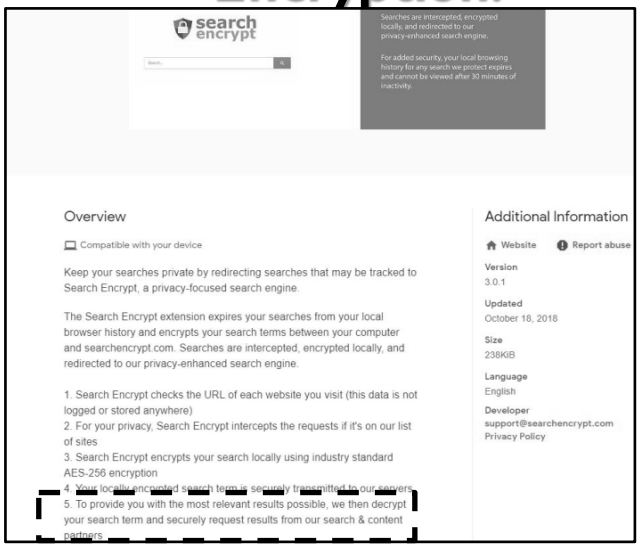


SEARCHABLE ENCRYPTION WITH FINE GRAINED ACCESS CONTROL

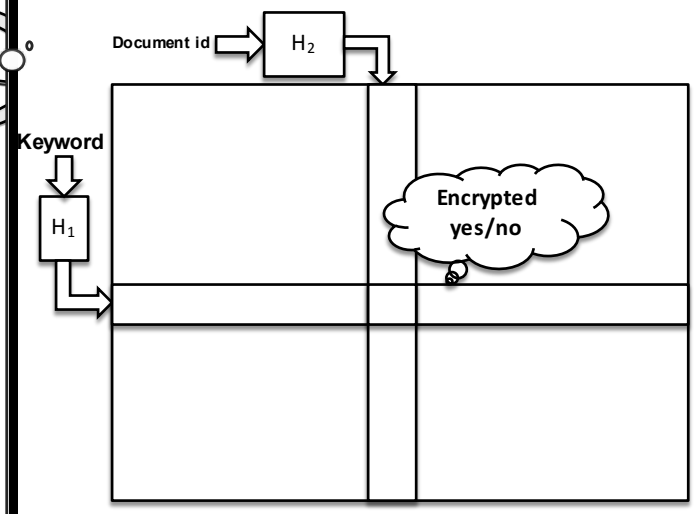
How Not to Do Searchable Encryption?



How Not to Do Searchable Encryption?



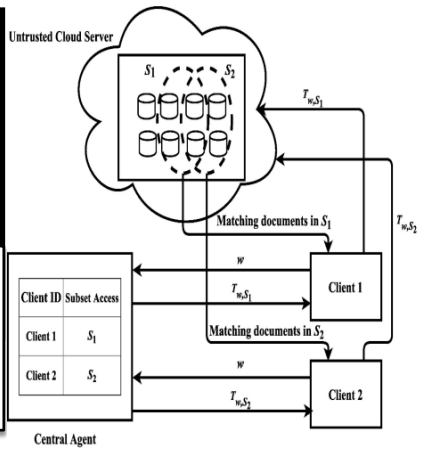
Searchable Index



CONTROLLED ACCESS SEARCHABLE ENCRYPTION (CASE)

- CASE inherently incorporates access control with security guarantees based on well known hardness assumptions
- CASE is scalable and resource-efficient in terms of storage on the cloud
- CASE is public key based with highly optimized key management techniques

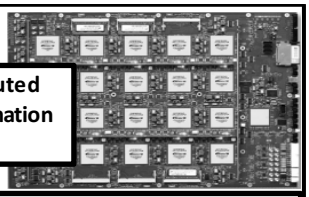
- The Cloud Server is assumed to be *honest but curious*
- What the Cloud Sees - *Encrypted Database* and *Search Tokens*
- What the Cloud does not learn - *Plaintext data* and *Search Queries*
- Security Guarantees:
  - Proven under standard cryptographic assumptions
  - Can be instantiated with elliptic curves.

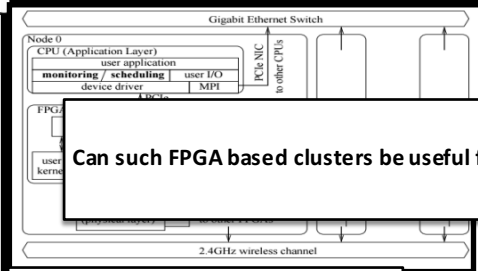


- Client has access to a specific subset of the overall database (in terms of document ids)
- The *search token* or *trapdoor* embeds this access control information and can only be used on the designated subset
- The trapdoor size does not depend on the size of the target subset and hence scales efficiently to arbitrarily large databases
- Can handle multiple clients in parallel

FPGAs for Distributed Applications

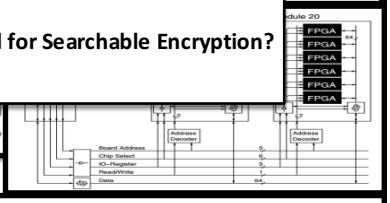
FPGA Clusters are being increasingly used for distributed applications in many different fields including information retrieval systems and cryptography





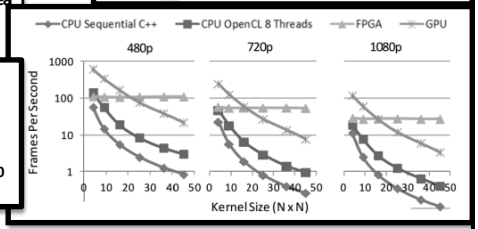
The COPACOBANA accelerator for cryptanalyzing block ciphers consists of 120 FPGAs

Can such FPGA based clusters be useful for Searchable Encryption?



FPGAs are even preferred over GPUs for distributed applications since they maintain real time performance over a wide range of inputs and operational conditions

Performance of a SAD (sum of absolute differences) application in software, GPU and FPGAs over several kernel configurations. FPGAs maintain real-time performance over the entire configuration range. Source: http://www.gstitt.ece.ufl.edu/courses/fall11/eel4930_5934/reading/sliding_window_fpga12.pdf



HARDWARE ACCELERATION FOR CASE USING FPGAS

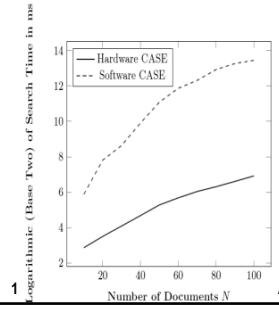
Hardware offers the advantage of massively parallel architectures to speed up searching on encrypted data

- A prototype implementation for CASE on multiple Artix 7 FPGAs
- Requires 192 FPGAs for a collection of 100 encrypted documents and a dictionary of 1000 keywords
- Affords better scalability and search performance as compared to software

Module	LUT Count	Register Count	No. of FPGAs Reqd.	Max. Freq. (MHz)
RAM	4522640	3999210	74	534.523
Setup	2769608	1335256	51	325.549
Keygen	272267	209160	2	325.464
BuildIndex	32124	12451	1	337.682
GenTrpdr	1884405	1118425	32	145.643
Search	2012945	1225671	32	138.765
Overall			192	> 100MHz

Post-Route Results: FPGA denominations

Scalability of the CASE Implementation



Hardware v/s Software: A Performance Comparison for the CASE Modules

Operation	Time (in ms) per Operation @100 MHz (Hardware)	Time (in ms) per Operation (Software)
RAM(Insert)	9.91×10^{-6}	6.89×10^{-5}
RAM(Delete)	9.91×10^{-6}	5.72×10^{-3}
RAM(Search)	9.91×10^{-6}	5.21×10^{-3}
RAM(Modify)	9.91×10^{-6}	5.72×10^{-3}
Setup	40.68	875.24
Keygen	6.05×10^{-1}	25.23
BuildIndex	1.21	57.21
GenTrpdr	65.86	9121.24
Search	121.32	11225.34

Result-Pattern Hiding Searchable Encryption for Conjunctive Queries

Joint work with:
Sikhar Patranabis (IIT Kharagpur)
Shangqi Lai, Amin Sakzad, Joseph Liu, Ron Steinfeld, Shifeng Sun, Dongxi Liu, Cong Zuo (Monash University)

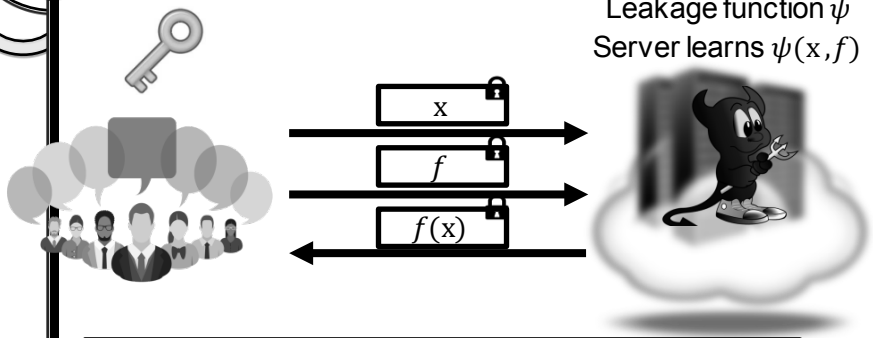
Sikhar Patranabis, Debdeep Mukhopadhyay: Lightweight Symmetric-Key Hidden Vector Encryption without Pairings. IACR Cryptology ePrint Archive 2017: 796 (2017)

Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shifeng Sun, Dongxi Liu, Cong Zuo: Result Pattern Hiding Searchable Encryption for Conjunctive Queries. CCS 2018: 745-762

Arnab Bag, Sikhar Patranabis, L Tribhuvan, Debdeep Mukhopadhyay, Hardware Acceleration for Searchable Encryption, Poster CCS 2018.

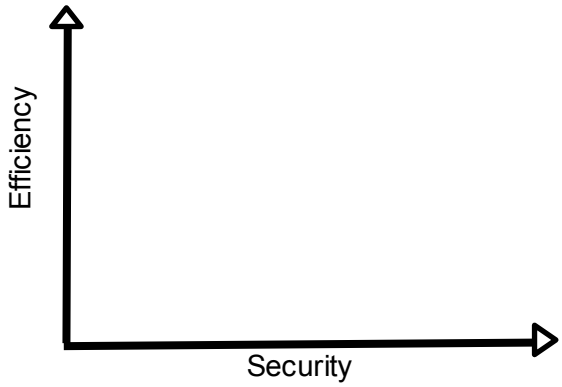
Searchable Encryption

Leakage function ψ
Server learns $\psi(x, f)$

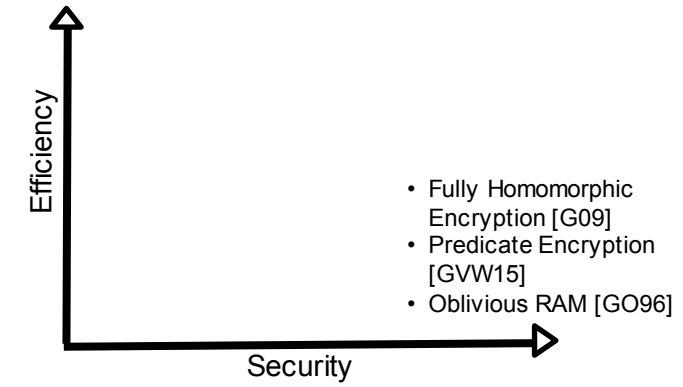


- **Security:** Minimize the information leakage to the server
- **Functionality:** Evaluate a rich class of functions on the encrypted data
- **Efficiency:** Reduce storage/time/communication overhead

Searchable Encryption: Compute on Encrypted Data

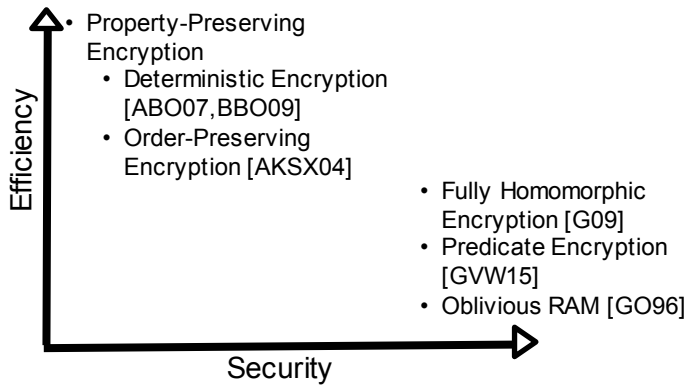


Searchable Encryption: Compute on Encrypted Data

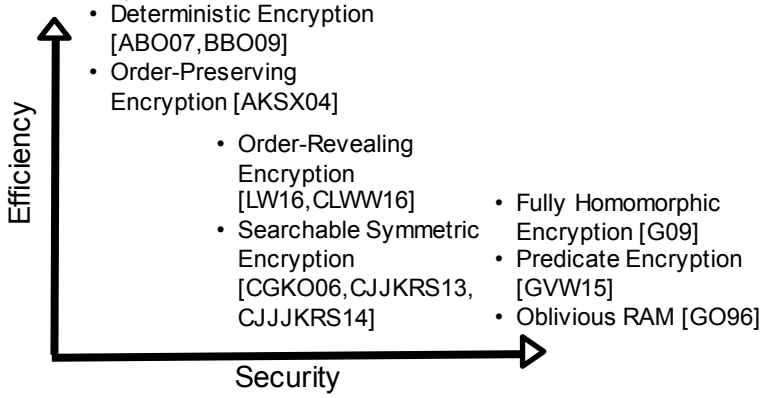


- Fully Homomorphic Encryption [G09]
- Predicate Encryption [GVW15]
- Oblivious RAM [GO96]

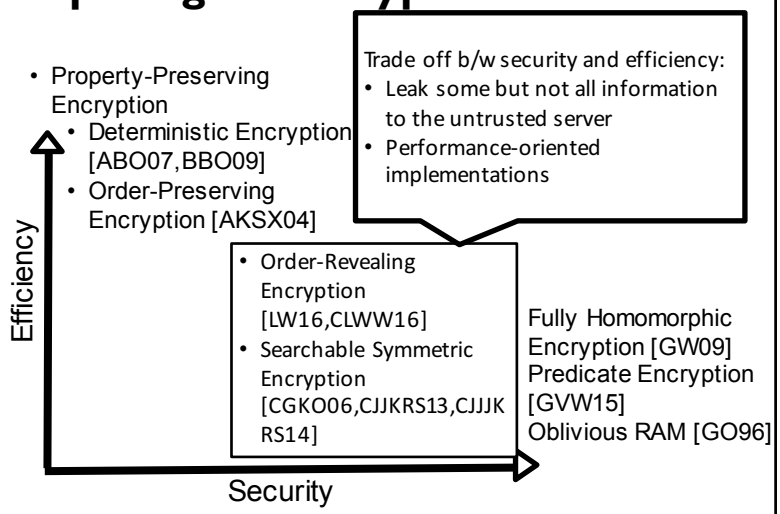
Computing on Encrypted Data



Computing on Encrypted Data



Computing on Encrypted Data



Searchable Symmetric Encryption

- Client holds a "small cryptographic key"
  - Initially:
    - Encrypts and outsources data to the server
    - Data pre-processed using efficient data structures
  - Later:
    - Uses the key to generate "encrypted query tokens"
    - Sends tokens to the server for query evaluation
    - Example query: *return all emails with "Alice" as recipient* ***and*** *"Bob" as sender* ***and*** *with "Crypto" in the subject*

Balancing Security and Efficiency: Existing Solutions

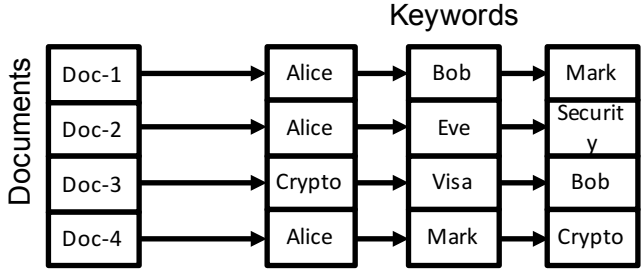
Searchable Symmetric Encryption
[CGKO06,CJJKRS13,CJJJKRS14]

- Client holds a "small cryptographic key"
  - Initially:
    - Encrypts and outsources data to the server
    - Data pre-processed using efficient data structures
  - Later:
    - Uses the key to generate "encrypted query tokens"
    - Sends tokens to the server for query evaluation

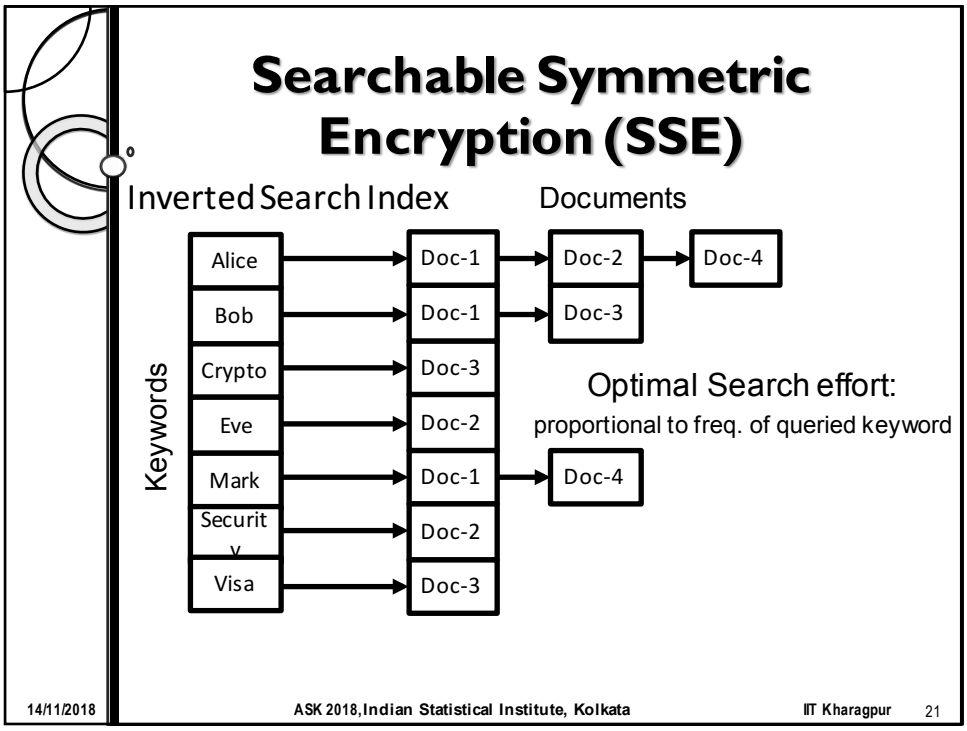
Single-reader single-writer framework

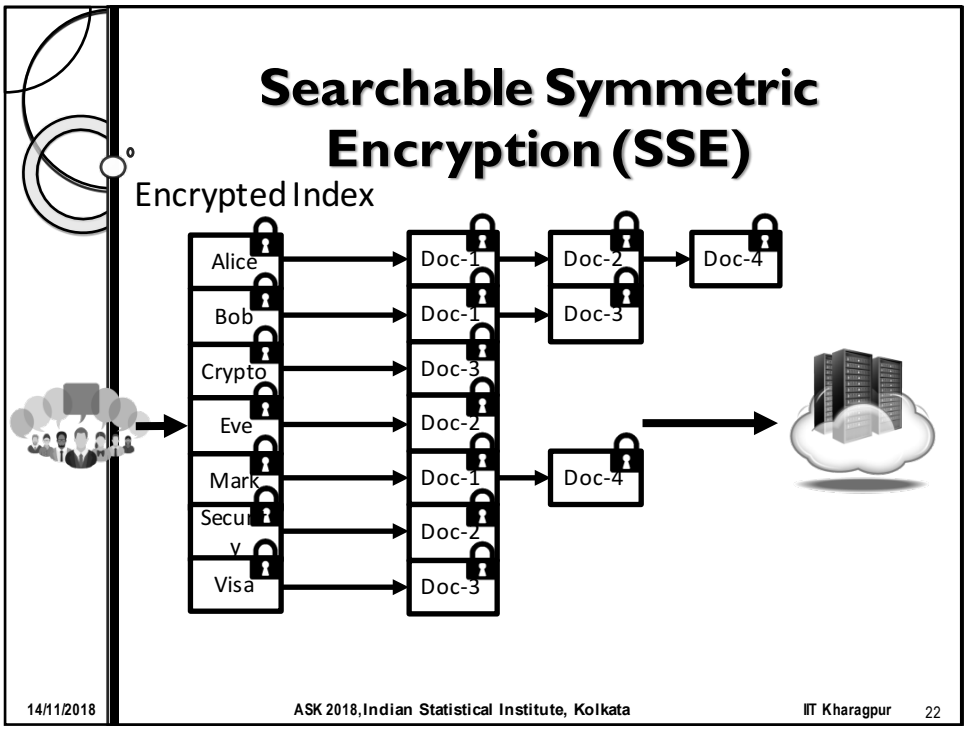
Searchable Symmetric Encryption (SSE)

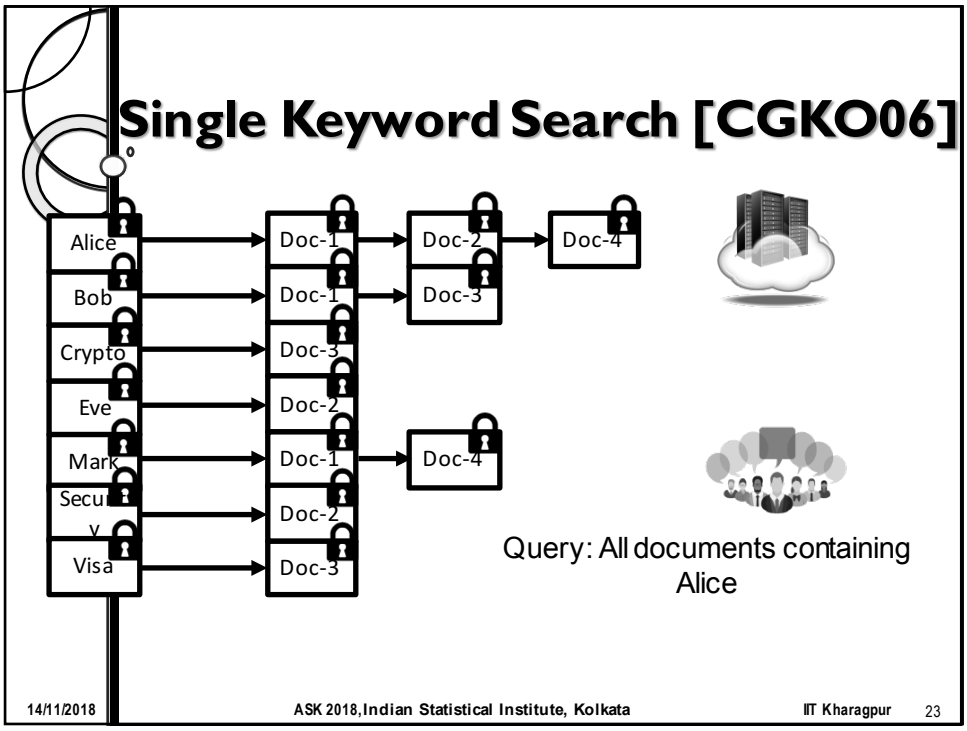
Data Structure: Search Index

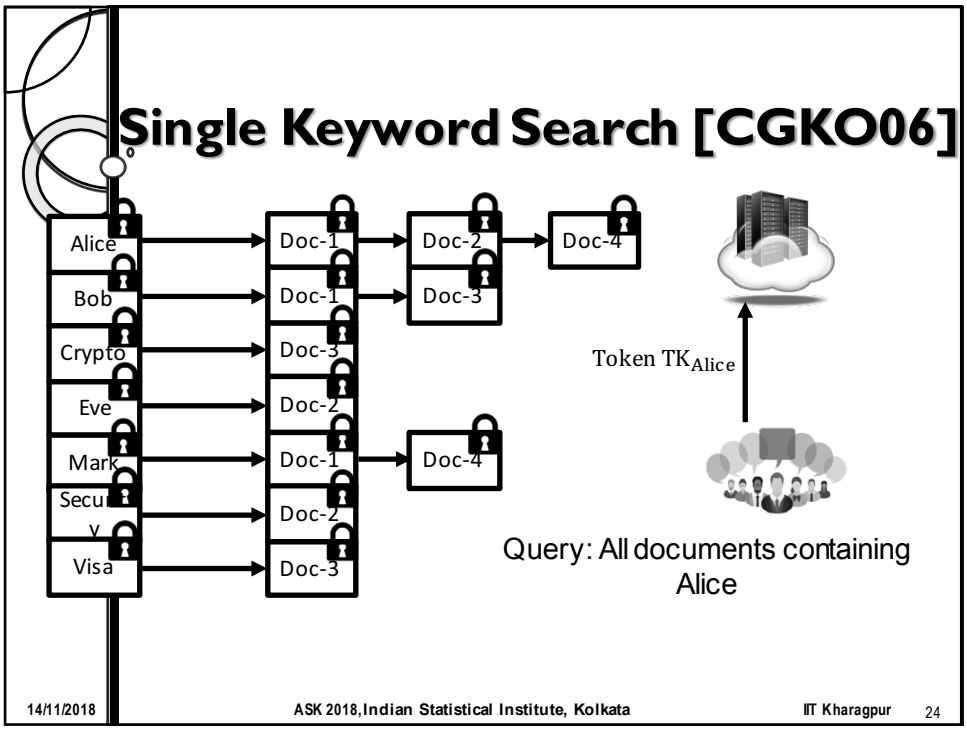


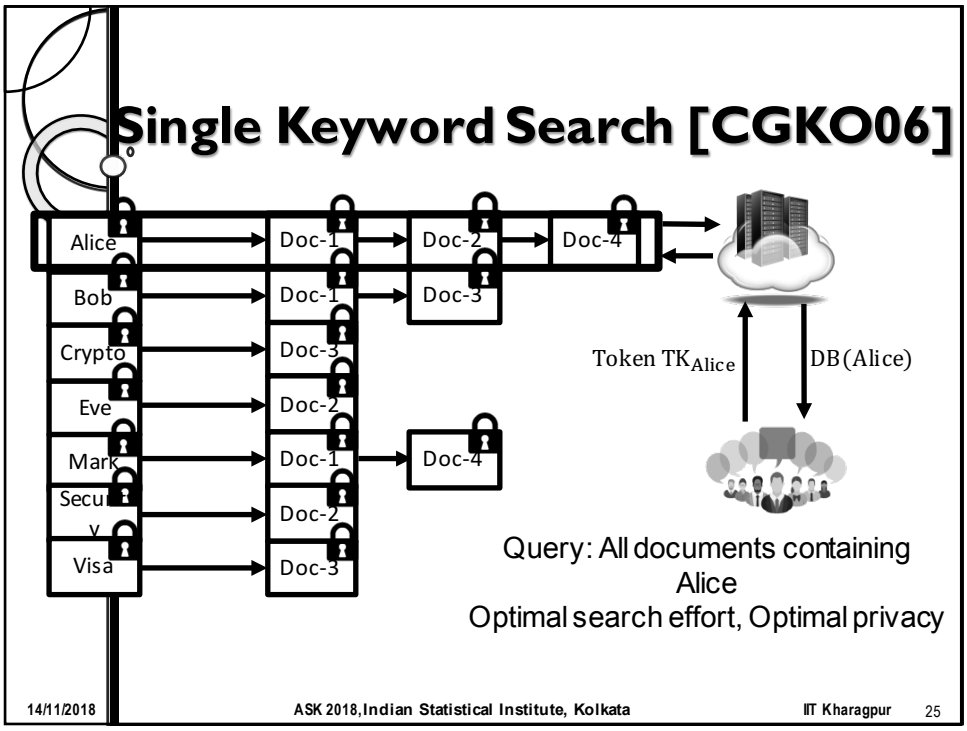
Non-optimal search effort: independent of freq. of keyword queried

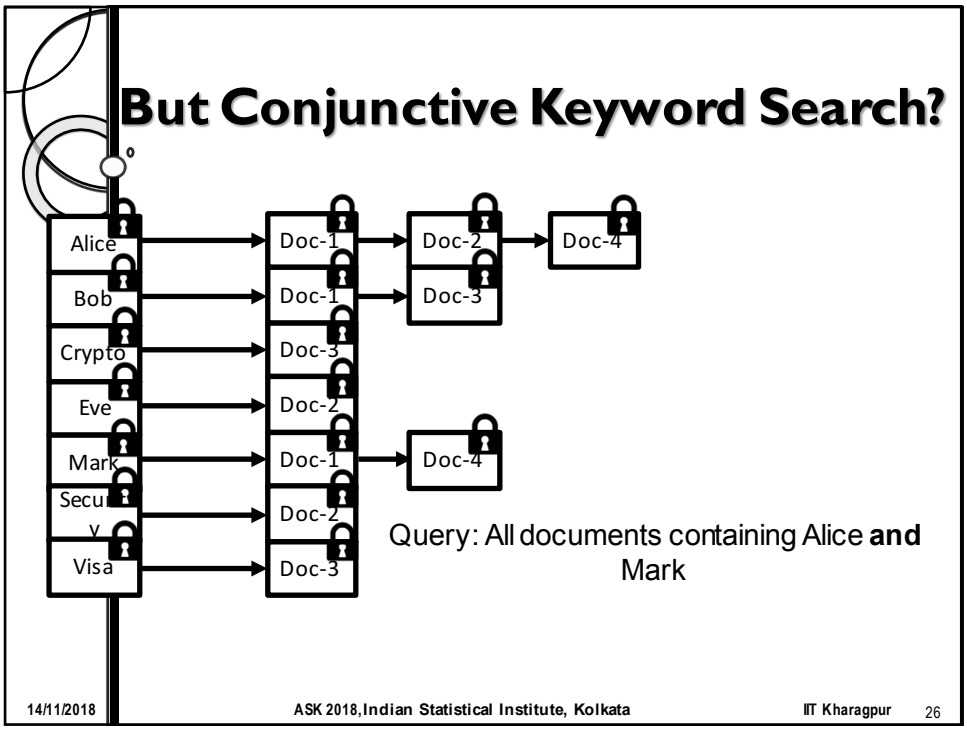

Searchable Symmetric Encryption (SSE)
Inverted Search Index
Documents
Keywords
Alice
Bob
Crypto
Eve
Mark
Securit y
Visa
Doc-1
Doc-1
Doc-3
Doc-2
Doc-1
Doc-2
Doc-3
Doc-2
Doc-3
Doc-4
Doc-4
Optimal Search effort:
proportional to freq. of queried keyword
14/11/2018
ASK 2018, Indian Statistical Institute, Kolkata
IIT Kharagpur
21

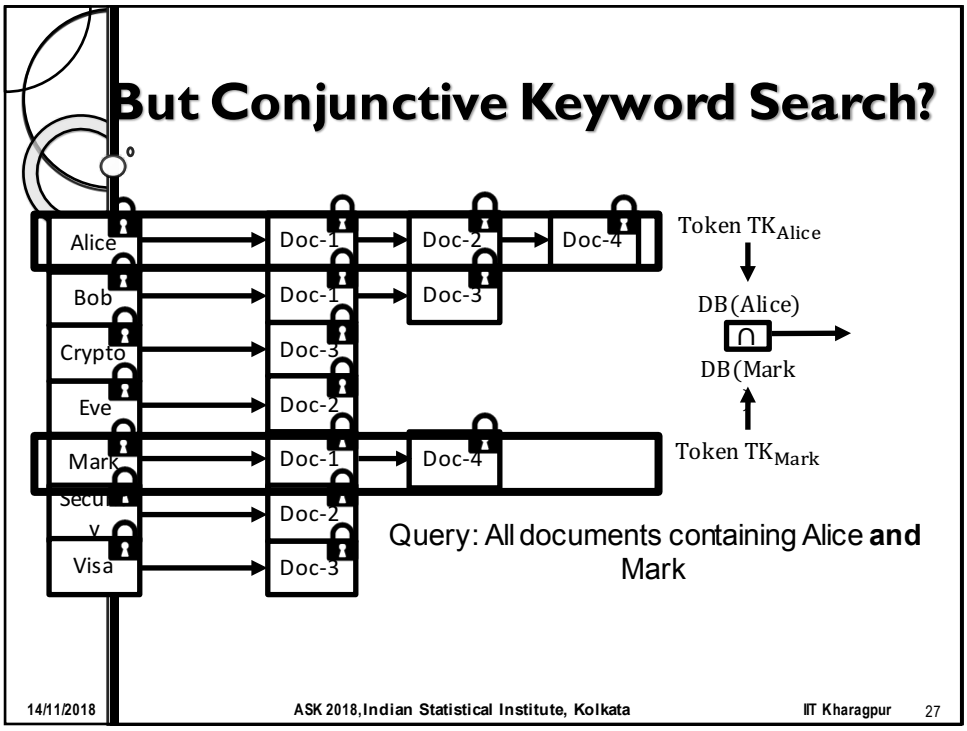

Searchable Symmetric Encryption (SSE)
Encrypted Index
Alice
Bob
Crypto
Eve
Mark
Secur y
Visa
Doc-1
Doc-1
Doc-3
Doc-2
Doc-1
Doc-2
Doc-3
Doc-2
Doc-3
Doc-4
Doc-4
14/11/2018
ASK 2018, Indian Statistical Institute, Kolkata
IIT Kharagpur
22

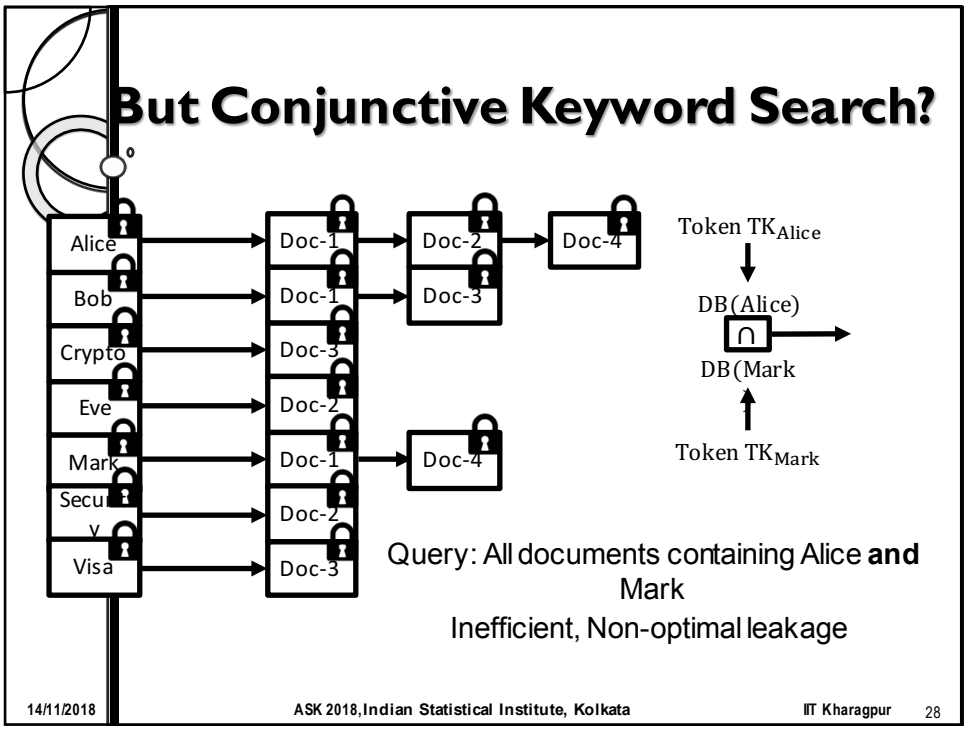

Single Keyword Search [CGKO06]
Alice
Bob
Crypto
Eve
Mark
Secur
y
Visa
Doc-1
Doc-1
Doc-3
Doc-2
Doc-1
Doc-2
Doc-3
Doc-2
Doc-3
Doc-4
Doc-4
Query: All documents containing Alice
14/11/2018
ASK 2018, Indian Statistical Institute, Kolkata
IIT Kharagpur
23


Single Keyword Search [CGKO06]
Alice
Bob
Crypto
Eve
Mark
Secur
y
Visa
Doc-1
Doc-1
Doc-3
Doc-2
Doc-1
Doc-2
Doc-3
Doc-2
Doc-3
Doc-4
Doc-4
Token TK_{Alice}
Query: All documents containing Alice
14/11/2018
ASK 2018, Indian Statistical Institute, Kolkata
IIT Kharagpur
24


Single Keyword Search [CGKO06]
Alice
Bob
Crypto
Eve
Mark
Secur
y
Visa
Doc-1
Doc-2
Doc-4
Doc-1
Doc-3
Doc-3
Doc-2
Doc-1
Doc-4
Doc-2
Doc-3
Token TK_{Alice}
DB(Alice)
Query: All documents containing Alice
Optimal search effort, Optimal privacy
14/11/2018
ASK 2018, Indian Statistical Institute, Kolkata
IIT Kharagpur
25


But Conjunctive Keyword Search?
Alice
Bob
Crypto
Eve
Mark
Secur
y
Visa
Doc-1
Doc-2
Doc-4
Doc-1
Doc-3
Doc-3
Doc-2
Doc-1
Doc-4
Doc-2
Doc-3
Query: All documents containing Alice **and** Mark
14/11/2018
ASK 2018, Indian Statistical Institute, Kolkata
IIT Kharagpur
26


But Conjunctive Keyword Search?
Alice
Bob
Crypto
Eve
Mark
Secur
y
Visa
Doc-1
Doc-2
Doc-4
Doc-1
Doc-3
Doc-3
Doc-2
Doc-1
Doc-4
Doc-2
Doc-3
Token TK$_{Alice}$
DB(Alice)
∩
DB(Mark
Token TK$_{Mark}$
Query: All documents containing Alice **and** Mark
14/11/2018
ASK 2018,Indian Statistical Institute, Kolkata
IIT Kharagpur
27


But Conjunctive Keyword Search?
Alice
Bob
Crypto
Eve
Mark
Secur
y
Visa
Doc-1
Doc-2
Doc-4
Doc-1
Doc-3
Doc-3
Doc-2
Doc-1
Doc-4
Doc-2
Doc-3
Token TK$_{Alice}$
DB(Alice)
∩
DB(Mark
Token TK$_{Mark}$
Query: All documents containing Alice **and** Mark
Inefficient, Non-optimal leakage
14/11/2018
ASK 2018,Indian Statistical Institute, Kolkata
IIT Kharagpur
28

Oblivious Cross Tags (OXT) [CJJKRS13]

Query: All documents containing keywords $w_1, w_2, \ldots, w_n$

Single-Keyword SSE

- Assume wlog that w_1 is the least frequent keyword or the s-term
- Perform single keyword-query on w_1 to recover $DB(w_1)$
- Suppose $DB(w_1) = (id_1, id_2, \ldots, id_m)$

Oblivious Cross Tags (OXT) [CJJKRS13]

Query: All documents containing keywords $w_1, w_2, \ldots, w_n$

Single-Keyword SSE

- Assume wlog that w_1 is the least frequent keyword or the s-term
- Perform single keyword-query on w_1 to recover $DB(w_1)$
- Suppose $DB(w_1) = (id_1, id_2, \ldots, id_m)$

Oracle

- Query with encrypted keyword $Enc(w_i)$ and encrypted doc-id $Enc(id_j)$
  - Returns 1 if $id_j \in DB(w_i)$
  - Returns 0 otherwise

Oblivious Cross Tags (OXT) [CJJKRS13]

- Query: All documents containing keywords $w_1, w_2, \ldots, w_n$

Single-Keyword SSE — S-Tag

- Assume wlog that w_1 is the least frequent keyword or the s-term
- Perform single keyword-query on w_1 to recover $DB(w_1)$
- Suppose $DB(w_1) = (id_1, id_2, \ldots, id_m)$

Oracle — Implemented as a two-party protocol — X-Tag

- Query with encrypted keyword $Enc(w_i)$ and encrypted doc-id $Enc(id_j)$
  - Returns 1 if $id_j \in DB(w_i)$
  - Returns 0 otherwise

Oblivious Cross Tags (OXT) [CJJKRS13]

- Query: All documents containing keywords $w_1, w_2, \ldots, w_n$

Single-Keyword SSE — S-Tag

- Assume wlog that w_1 is the least frequent keyword or the s-term
- Perform single keyword-query on w_1 to recover $DB(w_1)$
- Sup

Oracle

- Que
  doc
  - Returns 1 if $id_j \in DB(w_i)$
  - Returns 0 otherwise

- Search overhead for single-keyword SSE : $O(\sum_{i=1}^{n} |DB(w_i)|)$
- Search overhead for OXT: $O(|DB(w_1)| * n)$

Oblivious Cross Tags (OXT) [CJJKRS13]

- Query: All documents containing keywords $w_1, w_2, \ldots, w_n$

Single-Keyword SSE | S-Tag

- Assume wlog that w_1 is the least frequent keyword or the s-term
- Perform single keyword query on w_1 to recover $DB(w_1)$
- Suppose I

Oracle | Imp

- Query with doc-id $Enc(id_j)$
  - Returns 1 if $id_j \in DB(w_i)$
  - Returns 0 otherwise

Leakage: Keyword-pair result pattern $\{DB(w_1) \cap DB(w_i)\}_{i \in [n]}$

Vulnerable to **File Injection Attacks**

File Injection Attacks [CGPR15, ZKP16]

Encrypted files →

← Maliciously constructed files

Encrypted malicious files →

(Shown to be practical in [CGPR15])

File Injection Attacks [CGPR15, ZKP16]

Query-Recovery from Keyword-pair result pattern [ZKP16]

- Suppose W is the keyword-universe
- Inject L files each containing M random keywords from W
- Conjunctive query: $w_1 \wedge w_2 \wedge \cdots \wedge w_n$
  - Collect all injected files appearing in the outcome of the query
  - Take intersection of keywords in them

File Injection Attacks [CGPR15, ZKP16]

Query-Recovery from Keyword-pair result pattern [ZKP16]

- Suppose W is the keyword-universe
- Inject L files each containing M random keywords from W
- Conjunctive query: $w_1 \wedge w_2 \wedge \cdots \wedge w_n$
  - Collect all injected files appearing in the outcome of the query
  - Take intersection of keywords in them
- For L > 2nlog|W| and M = $(1/2)^{1/n}|W|$, with overwhelmingly large probability, this intersection *is* the set of queried keywords

File Injection Attacks [CGPR15, ZKP16]

Query-Recovery from Keyword-pair result pattern [ZKP16]

- Suppose W is the keyword-universe
- Inject L files each containing M random keywords from W
- Conjunctive query: $w_1 \wedge w_2 \wedge \cdots \wedge w_n$
  - Collect all injected files appearing in the outcome of the query
  - Take intersection of keywords in them
- For L > 2nlog|W| and M = $(1/2)^{1/n}|W|$, with overwhelmingly large probability, this intersection *is* the set of queried keywords
- Make n large for each query?

File Injection Attacks [CGPR15, ZKP16]

Query-Recovery from Keyword-pair result pattern [ZKP16]

- Suppose W is the keyword-universe
- Inject L files each containing M random keywords from W
- Conjunctive query: $w_1 \wedge w_2 \wedge \cdots \wedge w_n$
  - Collect all injected files appearing in the outcome of the query
  - Take intersection of keywords in them
- For L > 2nlog|W| and M = $(1/2)^{1/n}|W|$, with overwhelmingly large probability, this intersection *is* the set of queried keywords
- Make n large for each query?
  - Keyword-pair result pattern leakage➔ n is effectively 2 always
  - Attack becomes efficient and hard to circumvent

OUR AIM

- Design SSE scheme without keyword-pair result pattern leakage
  - Without compromising on efficiency
  - Without introducing additional leakages

Our Technique

Replace the X-Tag oracle in OXT by a different oracle that hides keyword-pair result pattern

Oracle-II	Implemented as a two-party protocol	H-Tag

- Query with a set of encrypted keywords $\{Enc(w_i)\}_{i\in[n]}$ and an encrypted document-identifier $Enc(id_j)$
  - Returns 1 if $id_j \in DB(w_i)$ for every $i \in [n]$
  - Returns 0 otherwise

Overview of Technique

Documents

Keywords	Doc-1	Doc-2	Doc-3	Doc-4	Doc-5
w_1	1	1	0	1	1
w_2	0	1	1	0	0
w_3	1	0	1	0	1
w_4	0	0	0	1	1
w_5	0	1	0	0	1

Query: $w_1 \wedge w_3 \wedge w_5$

Overview of Technique

- The least frequent keyword : w_5
  - Doc-ids: {2,5}

Documents

Keywords	Doc-1	Doc-2	Doc-3	Doc-4	Doc-5
w_1	1	1	0	1	1
w_2	0	1	1	0	0
w_3	1	0	1	0	1
w_4	0	0	0	1	1
w_5	0	1	0	0	1

Query: $w_1 \wedge w_3 \wedge w_5$

Overview of Technique

- The least frequent keyword : w_5
  - Doc-ids: {2,5}
- Create the following "query-table"

Documents

Keywords	Doc-1	Doc-2	Doc-3	Doc-4	Doc-5
w_1	1	1	0	1	1
w_2	0	1	1	0	0
w_3	1	0	1	0	1
w_4	0	0	0	1	1
w_5	0	1	0	0	1

	Doc-2	Doc-5
w_1	1	1
w_2	*	*
w_3	1	1
w_4	*	*
w_5	1	1

Query: $w_1 \wedge w_3 \wedge w_5$

Overview of Technique

- The least frequent keyword : w_5
  - Doc-ids: {2,5}
- Create the following "query-table"

Documents

Keywords	Doc-1	Doc-2	Doc-3	Doc-4	Doc-5
w_1	1	1	0	1	1
w_2	0	1	1	0	0
w_3	1	0	1	0	1
w_4	0	0	0	1	1
w_5	0	1	0	0	1

	Doc-2	Doc-5
w_1	1	1
w_2	*	*
w_3	1	1
w_4	*	*
w_5	1	1

- Match column-wise with the original table
  - Equality in all non-wildcard positions

Query: $w_1 \wedge w_3 \wedge w_5$

Overview of Technique

- The least frequent keyword : w_5
  - Doc-ids: {2,5}
- Create the following "query-table"

Documents

Keywords	Doc-1	Doc-2	Doc-3	Doc-4	Doc-5
w_1	1	1	0	1	1
w_2	0	1	1	0	0
w_3	1	0	1	0	1
w_4	0	0	0	1	1
w_5	0	1	0	0	1

	Doc-1	Doc-5
w_1	1	1
w_2	*	*
w_3	1	1
w_4	*	*
w_5	1	1

- Match column-wise with the original table
  - Equality in all non-wildcard positions

Query: $w_1 \wedge w_3 \wedge w_5$

Overview of Technique

Documents

Keywords	Doc-1	Doc-2	Doc-3	Doc-4	Doc-5
w_1	1	1	0	1	1
w_2	0	1	1	0	0
w_3	1	0	1	0	1
w_4	0	0	0	1	1
w_5	0	1	0	0	1

	Doc-1	Doc-5
w_1	1	1
w_2	*	*
w_3	1	1
w_4	1	1
w_5	1	1

Query: $w_1 \wedge w_3 \wedge w_4 \wedge w_5$

Overview of Technique

Documents

	Doc-1	Doc-2	Doc-3	Doc-4	Doc-5
w_1	1	1	0	1	1
w_2	0	1	1	0	0
w_3	1	0			
w_4	0	0	0	1	1
w_5	0	1	0	0	1

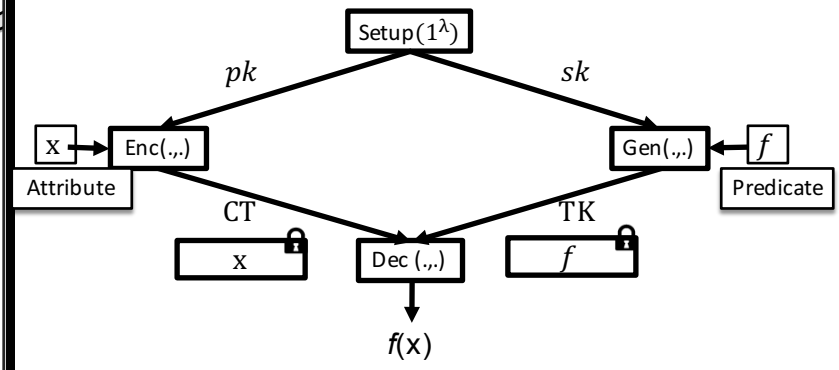
Keywords

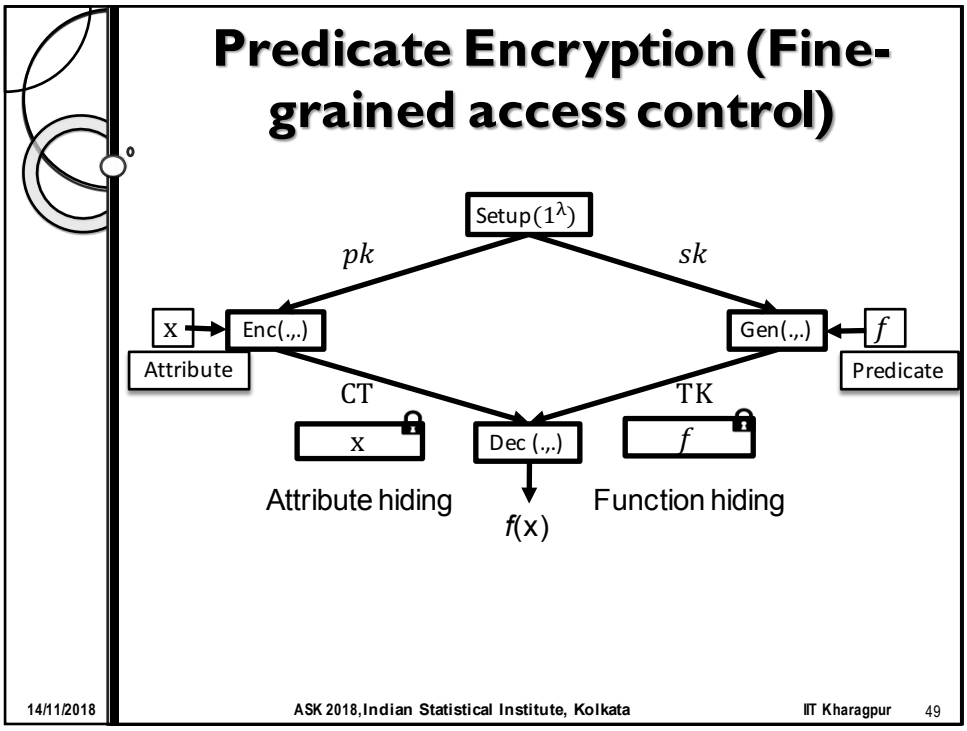
Hidden Vector Encryption [BW07]

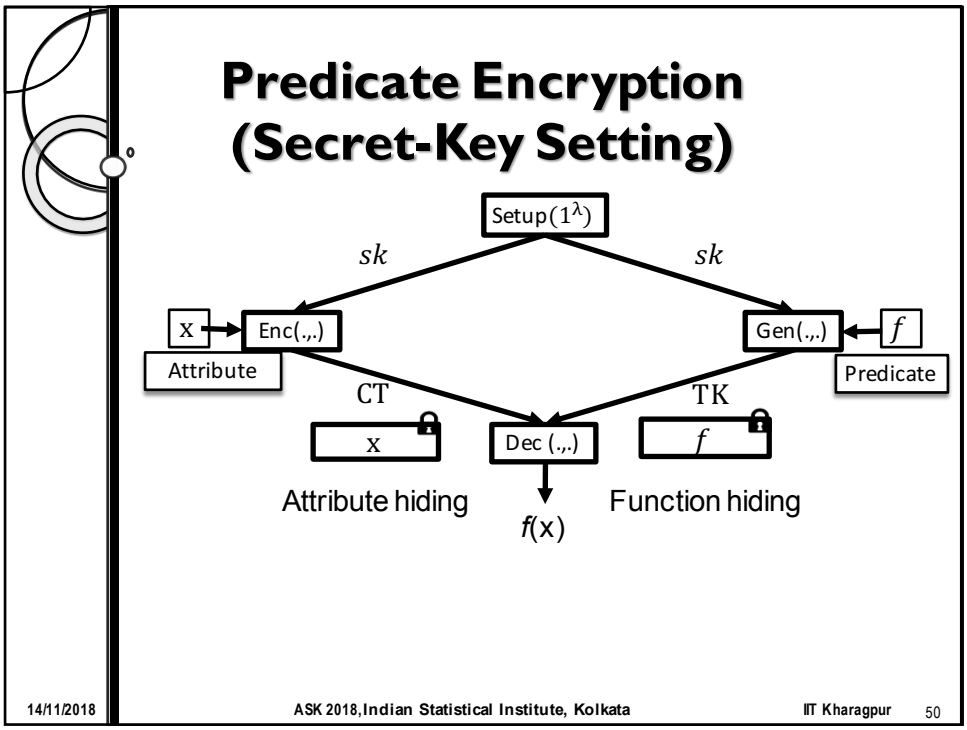
	Doc-1	Doc-5
w_1	1	1
w_2	*	*
		1
w_4	1	1
w_5	1	1

Query: $w_1 \wedge w_3 \wedge w_4 \wedge w_5$

Predicate Encryption (Fine-grained access control)




Predicate Encryption (Fine-grained access control)
Setup(1^λ)
pk
sk
x
Enc(.,.)
Attribute
Gen(.,.)
f
Predicate
CT
TK
x
Dec (.,.)
f
Attribute hiding
Function hiding
f(x)
14/11/2018
ASK 2018, Indian Statistical Institute, Kolkata
IIT Kharagpur
49


Predicate Encryption (Secret-Key Setting)
Setup(1^λ)
sk
sk
x
Enc(.,.)
Attribute
Gen(.,.)
f
Predicate
CT
TK
x
Dec (.,.)
f
Attribute hiding
Function hiding
f(x)
14/11/2018
ASK 2018, Indian Statistical Institute, Kolkata
IIT Kharagpur
50

Hidden Vector Encryption (a.k.a Wildcard IBE)

- A predicate encryption scheme over a finite alphabet $\Sigma \subset \{0,1\}^{\lambda}$ and a special wildcard symbol " $*$ "
- Each attribute is a vector with entries from Σ
  - $\vec{x} = (x_1, x_2, \ldots, x_n) \in \Sigma^n$
- Each predicate is defined with respect to a vector with entries from $\Sigma \cup \{*\}$
  - $\vec{v} = (v_1, v_2, \ldots, v_n) \in (\Sigma \cup \{*\})^n$
  - $f_{\vec{v}}(\vec{x}) = \begin{cases} 1, & (v_i = x_i) \vee (v_i = *) \text{ for each } i \in [n] \\ 0, & \text{otherwise} \end{cases}$

Hidden Vector Encryption: State-of-the-Art

Public-Key Constructions

- Based on either bilinear maps [BW07, KSW08, IP08, TT12] or lattices [AFV11]
- Attribute-hiding security against:
  - Polynomially many ciphertext queries
  - Polynomially many token-generation queries

Prototype implementation for vector length $n = 10^5$ (no wildcards)

Time (in secs)

Implementation		Enc	Gen	Dec
Public-Key	Bilinear Maps (prime order)	50.901	51.154	119.219
	Lattices (ring-LWE)	6.254	6.434	1.524

Hidden Vector Encryption: State-of-the-Art

Public-Key Constructions

- Based on either bilinear maps [BW07, KSW08, IP08, TT12] or lattices [AFV11]
- Attribute-hiding security against:
  - Polynomially many ciphertext queries
  - Polynomially many token-generation queries

Prototype implementation for vector length $n = 10^5$ (no wildcards)

Time (in secs)

Implementation		Enc	Gen	Test
Public-Key	Bilinear Maps (prime order)	50.901	51.154	119.219
	Lattices (ring-LWE)	6.254	6.434	1.524

Too inefficient for SSE-like applications

Hidden Vector Encryption: Our Contribution

- We construct the first symmetric-key HVE (SKHVE) scheme that:
  - Does not use bilinear maps/lattices
    - Can be implemented entirely using block ciphers in counter mode
  - Achieves attribute and function hiding security for:
    - A single ciphertext query (for the moment)
    - Polynomially many secret-key queries

Hidden Vector Encryption: Our Contribution

- We construct the first symmetric-key HVE (SKHVE) scheme that:
  - Does not use bilinear maps/lattices
    - Can be implemented entirely using block ciphers in counter mode
  - Achieves attribute and function hiding security for:
    - A single ciphertext query (for the moment)
    - Polynomially many secret-key queries
- We replace the X-Tag set in the OXT protocol with SKHVE ciphertexts
  - Retains non-interactivity (query-response still one communication round)
  - Imposes minimal overhead

Our SKHVE Construction

- Setup(1^λ): Output the description of:
  - PRF $F_k: \Sigma \times [n] \rightarrow \{0,1\}^\lambda$
  - SKE (G, E, D) with key and message space $\{0,1\}^\lambda$

Our SKHVE Construction

- Setup(1^λ): Output the description of:
  - PRF $F_k: \Sigma \times [n] \rightarrow \{0,1\}^\lambda$
  - SKE (G, E, D) with key and message space $\{0,1\}^\lambda$
- Enc$\left(k, \vec{x} = (x_1, \ldots, x_n)\right)$: Output $CT_{\vec{x}} = (F_k(x_1, 1), F_k(x_2, 2), \ldots, F_k(x_n, n))$

Our SKHVE Construction

- Setup(1^λ): Output the description of:
  - PRF $F_k: \Sigma \times [n] \rightarrow \{0,1\}^\lambda$
  - SKE (G, E, D) with key and message space $\{0,1\}^\lambda$
- Enc$\left(k, \vec{x} = (x_1, \ldots, x_n)\right)$: Output $CT_{\vec{x}} = (F_k(x_1, 1), F_k(x_2, 2), \ldots, F_k(x_n, n))$
- Gen$\left(k, \vec{v} = (v_1, \ldots, v_n)\right)$: Sample $k_j \leftarrow G(1^\lambda)$ for $j \in [n]$ and proceed as:
  - Set for each $j \in [n]$
    - $TK_j = \begin{cases} F_k(v_j, j) \oplus k_j, & v_j \neq * \\ \phi, & \text{otherwise} \end{cases}$, $k'_j = \begin{cases} k_j, & v_j \neq * \\ 0^\lambda, & \text{otherwise} \end{cases}$

Our SKHVE Construction

- Setup(1^λ): Output the description of:
  - PRF $F_k: \Sigma \times [n] \rightarrow \{0,1\}^\lambda$
  - SKE (G, E, D) with key and message space $\{0,1\}^\lambda$
- Enc$\left(k, \vec{x} = (x_1, \ldots, x_n)\right)$: Output $CT_{\vec{x}} = (F_k(x_1, 1), F_k(x_2, 2), \ldots, F_k(x_n, n))$
- Gen$\left(k, \vec{v} = (v_1, \ldots, v_n)\right)$: Sample $k_j \leftarrow G(1^\lambda)$ for $j \in [n]$ and proceed as:
  - Set for each $j \in [n]$
    - $TK_j = \begin{cases} F_k(v_j, j) \oplus k_j, & v_j \neq * \\ \phi, & \text{otherwise} \end{cases}$, $k'_j = \begin{cases} k_j, & v_j \neq * \\ 0^\lambda, & \text{otherwise} \end{cases}$
  - Set $TK_{n+1} = E((k'_1 \oplus k'_2 \oplus \cdots \oplus k'_n), 0^\lambda)$

Our SKHVE Construction

- Setup(1^λ): Output the description of:
  - PRF $F_k: \Sigma \times [n] \rightarrow \{0,1\}^\lambda$
  - SKE (G, E, D) with key and message space $\{0,1\}^\lambda$
- Enc$\left(k, \vec{x} = (x_1, \ldots, x_n)\right)$: Output $CT_{\vec{x}} = (F_k(x_1, 1), F_k(x_2, 2), \ldots, F_k(x_n, n))$
- Gen$\left(k, \vec{v} = (v_1, \ldots, v_n)\right)$: Sample $k_j \leftarrow G(1^\lambda)$ for $j \in [n]$ and proceed as:
  - Set for each $j \in [n]$
    - $TK_j = \begin{cases} F_k(v_j, j) \oplus k_j, & v_j \neq * \\ \phi, & \text{otherwise} \end{cases}$, $k'_j = \begin{cases} k_j, & v_j \neq * \\ 0^\lambda, & \text{otherwise} \end{cases}$
  - Set $TK_{n+1} = E((k'_1 \oplus k'_2 \oplus \cdots \oplus k'_n), 0^\lambda)$
  - Output $TK_{\vec{v}} = (TK_1, \ldots, TK_n, TK_{n+1})$

Our SKHVE Co

- Setup(1^λ): Output the descriptio
  - PRF $F_k: \Sigma \times [n] \rightarrow \{0,1\}^\lambda$
  - SKE (G, E, D) with key and message space $\{0,1\}^\lambda$
- Enc$\left(k, \vec{x} = (x_1, \ldots, x_n)\right)$: Output $CT_{\vec{x}} = (F_k(x_1, 1), F_k(x_2, 2), \ldots, F_k(x_n, n))$
- Gen$\left(k, \vec{v} = (v_1, \ldots, v_n)\right)$: Sample $k_j \leftarrow G(1^\lambda)$ for $j \in [n]$ and proceed as:
  - Set for each $j \in [n]$
    - $TK_j = \begin{cases} F_k(v_j, j) \oplus k_j, & v_j \neq * \\ \phi, & \text{otherwise} \end{cases}$, $k'_j = \begin{cases} k_j, & v_j \neq * \\ 0^\lambda, & \text{otherwise} \end{cases}$
  - Set $TK_{n+1} = E((k'_1 \oplus k'_2 \oplus \cdots \oplus k'_n), 0^\lambda)$
  - Output $TK_{\vec{v}} = (TK_1, \ldots, TK_n, TK_{n+1})$

- Dec$(CT_{\vec{x}}, TK_{\vec{v}})$:
  - Set $k'_j = \begin{cases} CT_j \oplus TK_j, & v_j \neq * \\ 0^\lambda, & \text{otherwise} \end{cases}$ for each $j \in [n]$
  - Compute $\delta = D((k'_1 \oplus k'_2 \oplus \cdots \oplus k'_n), TK_{n+1})$
    - If $\delta = 0^\lambda$, return 1
    - Else, return 0

Security

Simulation Experiment:

- Adversary chooses:
  - A challenge attribute vector $\vec{x}$
  - Polynomially many query vectors $\overrightarrow{v_1}, \ldots, \overrightarrow{v_q}$
- Given:
  - $f_{\overrightarrow{v_1}}(\vec{x}), \ldots, f_{\overrightarrow{v_q}}(\vec{x})$
  - Location of wildcard characters in $\overrightarrow{v_1}, \ldots, \overrightarrow{v_q}$
- Simulate:
  - The challenge ciphertext $CT_{\vec{x}}$
  - The tokens $TK_{\overrightarrow{v_1}}, \ldots, TK_{\overrightarrow{v_q}}$

Performance and Efficiency

Instantiation

- Use a block cipher in counter mode (e.g. AES-256)
  - Suffices for both PRF and CPA-secure PKE
  - Allows parallel processing

Performance and Efficiency

Instantiation

- Use a block cipher in counter mode (e.g. AES-256)
  - Suffices for both PRF and CPA-secure PKE
  - Allows parallel processing

Prototype implementation for vector length $n = 10^5$ (no wildcards)

Implementation

Time (in secs)

		Enc	Gen	Test
Public-Key	Bilinear Maps (prime order)	50.901	51.154	119.219
	Lattices (ring-LWE)	6.254	6.434	1.524
Secret-Key	Our SKHVE	0.162	0.172	0.004

Our SSE Scheme: The Overall Picture

Functionality

- Supports a variety of queries:
  - Conjunctive queries
  - Subset queries
  - Range queries

Not very efficient

Security

- Data is semantically secure
- Leaks:
  - File-access pattern (typically benign)
  - Single-keyword result pattern: $\mathrm{DB}(w_1)$
- Does not leak keyword-pair result pattern

Our SSE Scheme: The Overall Picture

Efficiency

- Preprocessing (encrypted index creation) time:
  - Scales linearly in size of database
- Overhead for n-keyword conjunctive/subset query:
  - One round non-interactive protocol using SKHVE.Gen and SKHVE.Test
  - Communication proportional to $|\mathrm{DB}(w_1)| * n$ where w_1 is the s-term
- Highly scalable; competitive with search on plaintext DB

Data Structures

- Should be I/O friendly
- Random access data structures:
  - Great for security
  - Kill scalability
- Use elliptic-curve based indexing for low latency [CJJKRS13]

Performance on Wikimedia Downloads

Size (GB)	#Documents	#Keywords	#Distinct (w,id) pairs
2.93	7.8×10^5	4.0×10^6	6.2×10^7
8.92	2.7×10^6	1.0×10^7	1.6×10^8
60.2	1.6×10^7	4.3×10^7	1.4×10^9

Performance on Wikimedia Downloads

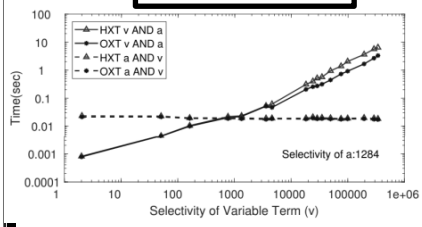
Size (GB)	#Documents	#Keywords	#Distinct (w,id) pairs
2.93	7.8×10^5	4.0×10^6	6.2×10^7
8.92	2.7×10^6	1.0×10^7	1.6×10^8
60.2	1.6×10^7	4.3×10^7	1.4×10^9

Query: recipient="Alice" and sender="Bob" and IP-addr = "69.89.31.226"
and
subject = "Crypto"
Query-response turnaround time: 0.85 sec

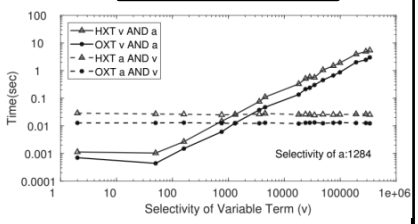
Performance on Wikimedia Downloads

Size (GB)	#Documents	#Keywords	#Distinct (w,id) pairs
2.93	7.8×10^5	4.0×10^6	6.2×10^7
8.92	2.7×10^6	1.0×10^7	1.6×10^8
60.2	1.6×10^7	4.3×10^7	1.4×10^9

Server Performance



Client Performance

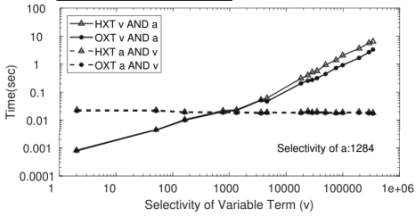


Performance on Wikimedia Downloads

Size (GB)	#Documents	#Keywords	#Distinct (w,id) pairs
2.93	7.8×10^5	4.0×10^6	6.2×10^7
8.92	2.7×10^6	1.0×10^7	1.6×10^8
60.2	1.6×10^7	4.3×10^7	1.4×10^9

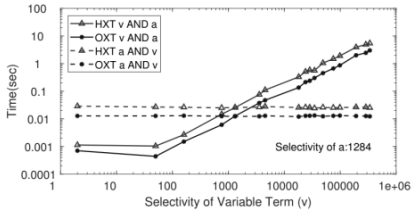
Server Performance

Degradation: 5-7%



Client Performance

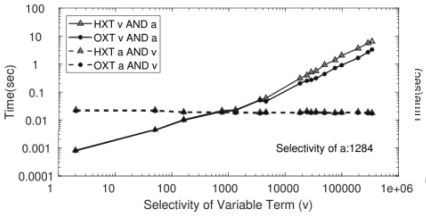
Degradation: 1.5-2x



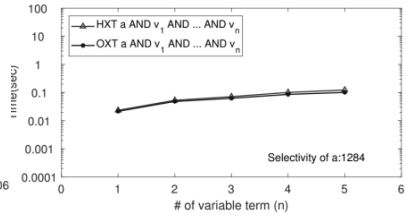
Performance on Wikimedia Downloads

Size (GB)	#Documents	#Keywords	#Distinct (w,id) pairs
2.93	7.8×10^5	4.0×10^6	6.2×10^7
8.92	2.7×10^6	1.0×10^7	1.6×10^8
60.2	1.6×10^7	4.3×10^7	1.4×10^9

Delay: 2-keyword queries



Delay: Multi-keyword queries



Can Hardware be leveraged to encrypted search less hard?

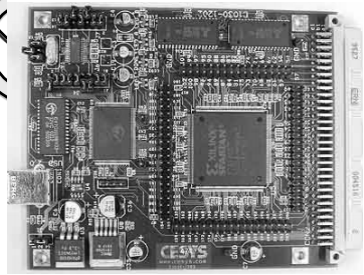
A Classic Example of a DES Cracker

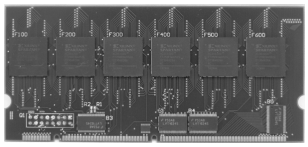
- In 1998, a custom hardware attack was mounted against the Data Encryption Standard :
  - $250,000 to build and decrypted DES cipher in 56 hours
- In 2006, COPACOBANA (Cost-Optimized PArallel COde Breaker) was built:
  - Consists of commercially available, reconfigurable integrated circuits.
  - $10,000 and decrypts DES cipher in around 6.4 days
  - Cost decrease by roughly a factor of 25
  - Adjusting for inflation over 8 years yields an even higher improvement of about 30x.
- Since 2007, SciEngines GmbH, a spin-off company of the two project partners of COPACOBANA has enhanced and developed successors of COPACOBANA.
  - In 2008 their COPACOBANA RIVYERA reduced the time to break DES to the current record of less than one day, using 128 Spartan-3 5000's

[Source: http://en.wikipedia.org/wiki/Custom_hardware_attack]

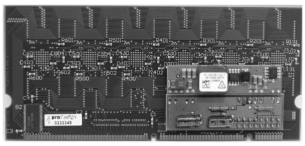
73

COPACOBANA Components

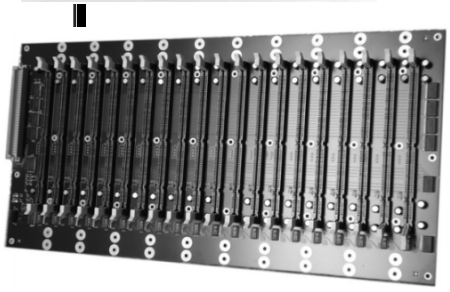




FPGA DIMM (front view)



FPGA DIMM and power module (rear view)

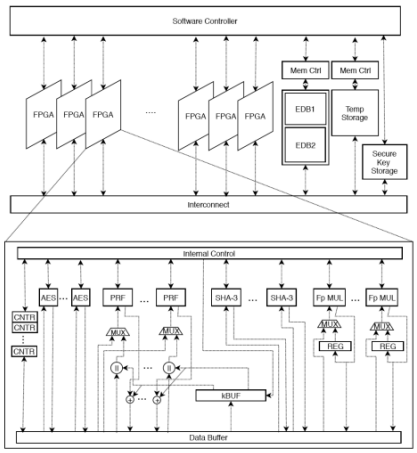


Plain Backplane

Source:
http://www.copacobana.org/paper/copacobana_gettingstarted.pdf

74

Hardware Architecture for SSE



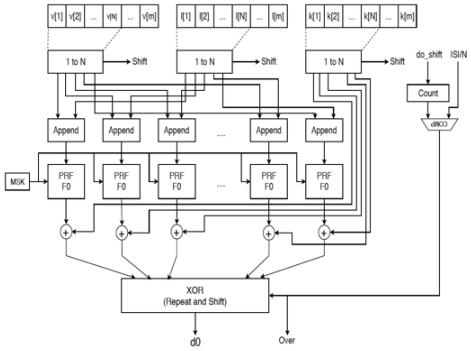
Architecture consists of a series of FPGAs, where each FPGA is programmed with RTL implementing the various cryptographic modules.

These modules are designed to operate in parallel on independent data chunks, made available via data buffer.
This is interfaced with the RAM and the memory controller.

Secret key is stored in a small tamper-proof non-volatile storage unit.

The software module facilitates the interaction between the hardware based crypto-accelerators and the RAM that stores the encrypted index of the SSE.

Parallelization in Hardware



GenToken Algorithm in our SSE:

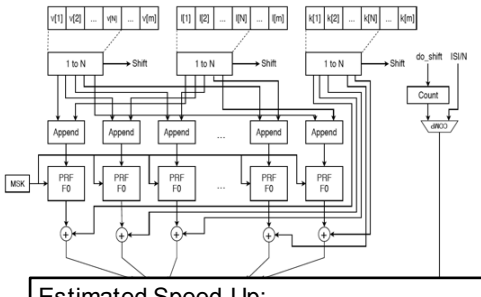
$$d_0 = \oplus_{j=1}^{n} (F_k(v_j, j) \oplus k_j)$$

The vectors v, I (holding the positions of non-wildcards), and k are realized as shift registers. Outputs are XORed using log(N) depth-tree of XORs.

Custer of FPGAs, PRFs and Enc as AES-256 hardware in counter-mode.
SHA-512 for hash function.
ECC is used for T-Set, using Curve25519

Primitive	Throughput(MBps)		
	Software	Hardware	Ratio
AES-256	693	21556	31.1x
SHA3-512	849	18768	22.1x
ECSM	0.16	2.16	13.5x

Parallelization in Hardware



GenToken Algorithm in our SSE:

$$d_0 = \bigoplus_{j=1}^{n} (F_k(v_j, j) \oplus k_j)$$

The vectors v, I (holding the positions of non-wildcards), and k are realized as shift registers. Outputs are XORed using log(N) depth-tree of XORs.

Estimated Speed-Up:
Pre-processing: 15x
Query-Response round: 20x

Custer of FPGAs, PRFs and Enc as AES-256 hardware in counter-mode.
SHA-512 for hash function.
ECC is used for T-Set, using Curve25519

Primitive	Throughput(MBps)		
	Software	Hardware	Ratio
AES-256	693	21556	31.1x
SHA3-512	849	18768	22.1x
ECSM	0.16	2.16	13.5x

Extensions and Open Problems

Support Updates
- Extend techniques from [CJJJKRS14] for updatable OXT
- But how to avoid increasing leakage?

Eliminate single-keyword result pattern leakage
- Inherent to the current design paradigm
- Might require new design paradigms to eliminate

Develop a complete FPGA prototype followed with a dedicated ASIC accelerator for SSE.

Thank you!

Source Code:
HVE: https://github.com/MonashCybersecurityLab/SHVE
HXT: https://github.com/MonashCybersecurityLab/HXT

8th International Conference on Security, Privacy, and Applied Cryptography Engineering

SPACE 2018

15-19 December 2018
Indian Institute of Technology, Kanpur

Submission Due : 20 July 2018, IST (GMT+5:30)
Notification : 31 August 2018
Final Version Due : 14 September 2018

Tutorials : 15 - 16 December 2018
Conference : 17 - 19 December 2018

The Internatio annual event
devoted to vari e conferences

SPACE 2018 is being organized by Indian Institute of Technology, Kanpur (IIT-K), in cooperation with International Association for Cryptologic Research (IACR) and Cryptology Research Society of India (CRSI) from 15 to 19 December, 2018. The program co-chairs for SPACE 2018 are Dr. Anupam Chattopadhyay (NTU Singapore), Dr. Chester Rebeiro (IIT Madras, India), and Dr. Yuval Yarom (The University of Adelaide and Data61, Australia).

The conference proceedings will be published by Springer as a Springer LNCS volume. The LNCS proceedings of the previous five years are available at SPACE 2017, SPACE 2016, SPACE 2015, SPACE 2014, SPACE 2013. The best contributions to SPACE 2018 will be selected for submission to regular issues of the Journal of Hardware and Systems Security (HASS).

Proceedings in Springer LNCS

In cooperation with

Please visit us at: https://space2018.cse.iitk.ac.in/