Searching on/Testing Encrypted Data

Lecture 23
Searchable Encryption
Searchable Encryption

A test key $T_w$ that allows one to test if $\text{Dec}_{SK}(C) = w$
Searchable Encryption

- A test key $T_w$ that allows one to test if $\text{Dec}_{SK}(C) = w$
- No other information about the message should be leaked
Searchable Encryption

- A test key $T_w$ that allows one to test if $\text{Dec}_{SK}(C) = w$

- No other information about the message should be leaked

- $w$ from a small dictionary of “keywords”
Searchable Encryption

- A test key $T_w$ that allows one to test if $\text{Dec}_{SK}(C) = w$
- No other information about the message should be leaked
- $w$ from a small dictionary of “keywords”

Public-Key Encryption with Keyword Search (PEKS)
Searchable Encryption

A test key $T_w$ that allows one to test if $\text{Dec}_{SK}(C) = w$

- No other information about the message should be leaked
- $w$ from a small dictionary of “keywords”

Public-Key Encryption with Keyword Search (PEKS)

- e.g. Application: delegating e-mail filtering
Searchable Encryption

- A test key $T_w$ that allows one to test if $\text{Dec}_{SK}(C) = w$
- No other information about the message should be leaked
- $w$ from a small dictionary of “keywords”

Public-Key Encryption with Keyword Search (PEKS)

- e.g. Application: delegating e-mail filtering

Sender attaches a list of (searchably) encrypted keywords to the (normally encrypted) mail. Receiver hands the mail-server test keys for keywords of its choice. Mail-server filters mails by checking for keywords and can forward them appropriately.
Searchable Encryption
Searchable Encryption

Components: $(PK, SK) \leftarrow \text{KeyGen}, T_w \leftarrow \text{TestKeyGen}(SK, w), Enc_{PK}(w), Dec_{SK}(C)$ and $Test_{T_w}(C)$
Searchable Encryption

Components: (PK, SK) ← KeyGen, Tw ← TestKeyGen(SK, w), EncPK(w), DecSK(C) and TestTw(C)

Correctness: For all (possibly adversarially chosen) words w, for C ← EncPK(w), we have DecSK(C) = w and TestTw(C) = 1. For any other (adversarially chosen) word w’, TestTw(C) = 0.
Searchable Encryption

Components: (PK, SK) ← KeyGen, \( T_w \leftarrow \text{TestKeyGen}(SK, w) \), Enc_{PK}(w), Dec_{SK}(C) and Test_{Tw}(C)

Correctness: For all (possibly adversarially chosen) words \( w \), for \( C \leftarrow \text{Enc}_{PK}(w) \), we have \( \text{Dec}_{SK}(C) = w \) and \( \text{Test}_{Tw}(C) = 1 \). For any other (adversarially chosen) word \( w' \), \( \text{Test}_{Tw}(C) = 0 \).

May require perfect or statistical correctness. Or, should hold w.h.p against computationally bounded environments choosing \( w, w' \) (after seeing PK, and for \( w' \), possibly after seeing \( C, T_w \) also).
Searchable Encryption

Components: \((PK, SK) \leftarrow \text{KeyGen}, T_w \leftarrow \text{TestKeyGen}(SK, w), \text{Enc}_{PK}(w), \text{Dec}_{SK}(C)\) and \(\text{Test}_{Tw}(C)\)

Correctness: For all (possibly adversarially chosen) words \(w\), for \(C \leftarrow \text{Enc}_{PK}(w)\), we have \(\text{Dec}_{SK}(C) = w\) and \(\text{Test}_{Tw}(C) = 1\). For any other (adversarially chosen) word \(w'\), \(\text{Test}_{Tw}(C) = 0\).

May require perfect or statistical correctness. Or, should hold \(w.h.p\) against computationally bounded environments choosing \(w, w'\) (after seeing \(PK\), and for \(w'\), possibly after seeing \(C, Tw\) also).

Secrecy: CPA or CCA security against adversary with oracle access to \(\text{TestKeyGen}(SK, .)\), as long as adversary doesn’t query \(w_0, w_1\)
Trivial Solution: using PKE
Trivial Solution: using PKE

If the dictionary is small, \((PK, SK) = \{(PK_w, SK_w) \mid w \text{ in dictionary}\}\)
Trivial Solution: using PKE

- If the dictionary is small, \((PK, SK) = \{(PK_w, SK_w) \mid w \text{ in dictionary}\}\)
- To encrypt a keyword (or, in fact, a list of keywords), \(Enc_{PK}(w) = \langle Enc_{PK_1}(0), ..., Enc_{PK_w}(1), ..., Enc_{PK_n}(0)\rangle\)
Trivial Solution: using PKE

If the dictionary is small, \((PK, SK) = \{ (PK_w, SK_w) \mid w \text{ in dictionary} \}\)

To encrypt a keyword (or, in fact, a list of keywords), \(Enc_{PK}(w) = \langle Enc_{PK_1}(0), \ldots, Enc_{PK_w}(1), \ldots, Enc_{PK_n}(0) \rangle\)

\(TestKeyGen(SK, w) = SK_w\)
Trivial Solution: using PKE

- If the dictionary is small, \((PK, SK) = \{ (PK_w, SK_w) \mid w \text{ in dictionary} \}\)

- To encrypt a keyword (or, in fact, a list of keywords), \(Enc_{PK}(w) = \langle Enc_{PK_1}(0), \ldots, Enc_{PK_w}(1), \ldots, Enc_{PK_n}(0) \rangle\)

- \(TestKeyGen(SK, w) = SK_w\)

- Keys and ciphertexts proportional to the dictionary size
Trivial Solution: using IBE
Trivial Solution: using IBE

Derive $(PK_w, SK_w)$ as keys in an IBE scheme for identity $w$
Trivial Solution: using IBE

- Derive $\mathbf{(PK_w, SK_w)}$ as keys in an IBE scheme for identity $w$
- $(PK, SK) = (MPK, MSK)$ (master keys) for the IBE
Trivial Solution: using IBE

Derive \((PK_w, SK_w)\) as keys in an IBE scheme for identity \(w\)

\((PK, SK) = (MPK, MSK)\) (master keys) for the IBE

To encrypt a keyword (or, in fact, a list of keywords), 
\[\text{Enc}_{PK}(w) = \langle \text{IBEnc}_{PK}(0; id=0), \ldots, \text{IBEnc}_{PK}(1; id=w), \ldots, \text{IBEnc}_{PK}(0; id=n)\rangle\]
Derive \((PK_w, SK_w)\) as keys in an IBE scheme for identity \(w\)

\((PK, SK) = (MPK, MSK)\) (master keys) for the IBE

To encrypt a keyword (or, in fact, a list of keywords), \(Enc_{PK}(w) = <IBEnc_{PK}(0;id=0), ..., IBEnc_{PK}(1;id=w), ..., IBEnc_{PK}(0;id=n)>\)

\(TestKeyGen(SK,w) = SK_w\), the secret-key for \(id=w\)
Trivial Solution: using IBE

- Derive \((PK_w, SK_w)\) as keys in an IBE scheme for identity \(w\)

- \((PK, SK) = (MPK, MSK)\) (master keys) for the IBE

- To encrypt a keyword (or, in fact, a list of keywords), \(Enc_{PK}(w) = <IBEnc_{PK}(0; id=0), ..., IBEnc_{PK}(1; id=w), ..., IBEnc_{PK}(0; id=n)>\)

- \(TestKeyGen(SK, w) = SK_w\), the secret-key for \(id=w\)

- Compact keys, but ciphertext is still long
PEKS from Anonymous IBE
PEKS from Anonymous IBE

Suppose, to encrypt a keyword $\text{Enc}_{PK}(w) = \text{IBE}_{\text{Enc}}_{PK}(1; id=w)$
PEKS from Anonymous IBE

Suppose, to encrypt a keyword $\text{Enc}_{PK}(w) = \text{IBE}\text{nc}_{PK}(1; id=w)$

Secure?
PEKS from Anonymous IBE

Suppose, to encrypt a keyword $\text{Enc}_{PK}(w) = \text{IBEnc}_{PK}(1;id=w)$

Secure?

IBE ciphertexts may reveal id (can have the id in the clear)
Suppose, to encrypt a keyword $Enc_{PK}(w) = \text{IBE}Enc_{PK}(1;id=w)$

Secure?

IBE ciphertexts may reveal id (can have the id in the clear)

Anonymous IBE
Suppose, to encrypt a keyword $\text{Enc}_{\text{PK}}(w) = \text{IBEnc}_{\text{PK}}(1; \text{id}=w)$

Secure?

IBE ciphertexts may reveal id (can have the id in the clear)

Anonymous IBE

Ciphertext does not reveal id used, unless has key for that id
**PEKS from Anonymous IBE**

- Suppose, to encrypt a keyword $\text{Enc}_{PK}(w) = \text{IBEEnc}_{PK}(1; id=w)$
  
  - Secure?
  
  - IBE ciphertexts may reveal id (can have the id in the clear)

- Anonymous IBE
  
  - Ciphertext does not reveal id used, unless has key for that id
  
  - cf. Anonymous (or key-private) encryption: ciphertext does not reveal the PK used for encryption (unless SK known)
PEKS from Anonymous IBE

Suppose, to encrypt a keyword $\text{Enc}_{\text{PK}}(w) = \text{IBEnc}_{\text{PK}}(1; \text{id}=w)$

Secure?

IBE ciphertexts may reveal id (can have the id in the clear)

Anonymous IBE

Ciphertext does not reveal id used, unless has key for that id

cf. Anonymous (or key-private) encryption: ciphertext does not reveal the PK used for encryption (unless SK known)

Consistency issue: IBE makes no guarantees about what the output is when decrypted using a wrong id’s key (except that it reveals nothing about the correct plaintext)
PEKS from Anonymous IBE
PEKS from Anonymous IBE

Consistency issue: IBE makes no guarantees about what the output is when decrypted using a wrong id’s key (except that it reveals nothing about the correct plaintext)
PEKS from Anonymous IBE

Consistency issue: IBE makes no guarantees about what the output is when decrypted using a wrong id’s key (except that it reveals nothing about the correct plaintext)

To encrypt a keyword, $\text{Enc}_{PK}(w) = (\text{IBE} \text{Enc}_{PK}(r;\text{id}=w), r)$ for a random message $r$ ($|r|=k$)
PEKS from Anonymous IBE

Consistency issue: IBE makes no guarantees about what the output is when decrypted using a wrong id’s key (except that it reveals nothing about the correct plaintext)

To encrypt a keyword, $\text{Enc}_{PK}(w) = (\text{IBEnc}_{PK}(r;id=w), r)$ for a random message $r$ ($|r|=k$)

If decrypting $\text{IBEnc}_{PK}(r;id=w)$, for a random $r$, using a wrong id’s key gives $r$ with significant probability, then breaks IBE security
PEKS from Anonymous IBE

Consistency issue: IBE makes no guarantees about what the output is when decrypted using a wrong id’s key (except that it reveals nothing about the correct plaintext)

To encrypt a keyword, $\text{Enc}_{PK}(w) = (\text{IBEnc}_{PK}(r;id=w), r)$ for a random message $r$ ($|r|=k$)

If decrypting $\text{IBEnc}_{PK}(r;id=w)$, for a random $r$, using a wrong id’s key gives $r$ with significant probability, then breaks IBE security

Breaking IBE’s security: give out $r_0,r_1$; decrypt challenge using the wrong id’s key; probability of getting $r_0$ when encryption is of $r_1$ is $2^{-k}$, but is significant when it is of $r_0$
PEKS from Anonymous IBE

- Consistency issue: IBE makes no guarantees about what the output is when decrypted using a wrong id's key (except that it reveals nothing about the correct plaintext)

- To encrypt a keyword, $\text{Enc}_{PK}(w) = (\text{IBE}\text{Enc}_{PK}(r;id=w), r)$ for a random message $r$ ($|r|=k$)

- If decrypting $\text{IBE}\text{Enc}_{PK}(r;id=w)$, for a random $r$, using a wrong id's key gives $r$ with significant probability, then breaks IBE security

- Breaking IBE's security: give out $r_0, r_1$; decrypt challenge using the wrong id's key; probability of getting $r_0$ when encryption is of $r_1$ is $2^{-k}$, but is significant when it is of $r_0$

- Or add such “decryption recognition” directly to Anonymous IBE
Predicate Encryption
Predicate Encryption

Test for properties of encrypted attributes
Predicate Encryption

Test for properties of encrypted attributes

For $C \leftarrow \text{Enc}_{PK}(a)$, we require that boolean $\text{Test}_{Tp}(C) = 1$ iff $P(a) = 1$
Predicate Encryption

Test for properties of encrypted attributes

For $C \leftarrow \text{Enc}_{PK}(a)$, we require that boolean $\text{Test}_{TP}(C) = 1$ iff $P(a) = 1$
Predicate Encryption

- Test for properties of encrypted attributes
- For $C \leftarrow \text{Enc}_{\text{PK}}(a)$, we require that boolean $\text{Test}_{T_P}(C)=1$ iff $P(a)=1$
  - Or $\text{Test}_{T_P}(C) = P(a)$, for a function $P$ (e.g. $P(a,m)=m$ if $P'(a)=1$, else ⊥)

$T_P$ is the key to test for property $P$
Predicate Encryption

- Test for properties of encrypted attributes
  - For $C \leftarrow \text{Enc}_{PK}(a)$, we require that boolean $\text{Test}_{T_P}(C)=1$ iff $P(a)=1$
  - Or $\text{Test}_{T_P}(C) = P(a)$, for a function $P$ (e.g. $P(a,m)=m$ if $P'(a)=1$, else ⊥)
  - $P$ from a certain predicate family will be supported

$T_P$ is the key to test for property $P$
Predicate Encryption

- Test for properties of encrypted attributes

For $C \leftarrow \text{Enc}_{PK}(a)$, we require that boolean $\text{Test}_{Tp}(C) = 1$ iff $P(a) = 1$

- Or $\text{Test}_{Tp}(C) = P(a)$, for a function $P$ (e.g. $P(a, m) = m$ if $P'(a) = 1$, else ⊥)

$P$ from a certain predicate family will be supported

- e.g. $P$ that checks for equality ($a = w$?) (i.e., PEKS), or for range ($a \in [r, s]$?) or membership in a list ($a \in S$?)

$T_p$ is the key to test for property $P$
Predicate Encryption

Test for properties of encrypted attributes

For \( C \leftarrow \text{Enc}_{PK}(a) \), we require that boolean \( \text{Test}_{TP}(C) = 1 \) iff \( P(a) = 1 \)

Or \( \text{Test}_{TP}(C) = P(a) \), for a function \( P \) (e.g. \( P(a,m) = m \) if \( P'(a) = 1 \), else \( \bot \))

\( P \) from a certain predicate family will be supported

e.g. \( P \) that checks for equality \( (a=w?) \) (i.e., PEKS), or for range \( (a \in [r,s]?) \) or membership in a list \( (a \in S?) \)

Trivial solution, when the predicate family is small

\( TP \) is the key to test for property \( P \)
Predicate Encryption

- Test for properties of encrypted attributes
  - For $C \leftarrow \text{Enc}_{PK}(a)$, we require that boolean $\text{Test}_{Tp}(C) = 1$ iff $P(a) = 1$
  - Or $\text{Test}_{Tp}(C) = P(a)$, for a function $P$ (e.g. $P(a,m) = m$ if $P'(a) = 1$, else ⊥)

- $P$ from a certain predicate family will be supported
  - e.g. $P$ that checks for equality ($a = w?$) (i.e., PEKS), or for range ($a \in [r,s]?$) or membership in a list ($a \in S?$)

- Trivial solution, when the predicate family is small
  - $(PK, SK) = \{(PK_P, SK_P) \mid P \text{ in the predicate family}\}$. Ciphertext has $\text{Enc}_{PK_p}(P(a))$ for each $P$. 
  
  $T_P$ is the key to test for property $P$
Predicate Encryption

Test for properties of encrypted attributes

For $C \leftarrow \text{Enc}_{PK}(a)$, we require that boolean $\text{Test}_{Tp}(C) = 1$ iff $P(a) = 1$

Or $\text{Test}_{Tp}(C) = P(a)$, for a function $P$ (e.g. $P(a, m) = m$ if $P'(a) = 1$, else $\perp$)

$P$ from a certain predicate family will be supported

e.g. $P$ that checks for equality ($a=w?$) (i.e., PEKS), or for range ($a \in [r, s]?$) or membership in a list ($a \in S?$)

Trivial solution, when the predicate family is small

$(PK, SK) = \{(PK_P, SK_P) \mid P \text{ in the predicate family}\}$. Ciphertext has $\text{Enc}_{PK_p}(P(a))$ for each $P$.

Can support functions instead of predicates
Predicate Encryption

Test for properties of encrypted attributes

For $C \leftarrow \text{Enc}_{PK}(a)$, we require that boolean $\text{Test}_{T_P}(C) = 1$ iff $P(a) = 1$

Or $\text{Test}_{T_P}(C) = P(a)$, for a function $P$ (e.g. $P(a,m) = m$ if $P'(a) = 1$, else $\perp$)

$P$ from a certain predicate family will be supported

e.g. $P$ that checks for equality ($a = w\,?$) (i.e., PEKS), or for range ($a \in [r,s]\,?$) or membership in a list ($a \in S\,?$)

Trivial solution, when the predicate family is small

$(PK,SK) = \{(PK_P,SK_P) \mid P \text{ in the predicate family}\}$. Ciphertext has $\text{Enc}_{PK_P}(P(a))$ for each $P$.

Can support functions instead of predicates

e.g. Can attach a message to be revealed if Test positive
Predicate Encryption

Test for properties of encrypted attributes

For \( C \leftarrow \text{Enc}_{PK}(a) \), we require that boolean \( \text{Test}_{T_P}(C) = 1 \) iff \( P(a) = 1 \)

Or \( \text{Test}_{T_P}(C) = P(a) \), for a function \( P \) (e.g. \( P(a,m) = m \) if \( P'(a) = 1 \), else \( \bot \))

\( P \) from a certain predicate family will be supported

e.g. \( P \) that checks for equality \( (a=w?) \) (i.e., PEKS), or for range \( (a \in [r,s]?) \) or membership in a list \( (a \in S?) \)

Trivial solution, when the predicate family is small

\( (PK,SK) = \{(PK_P,SK_P) \mid P \text{ in the predicate family}\} \). Ciphertext has \( \text{Enc}_{PK_P}(P(a)) \) for each \( P \).

Can support functions instead of predicates

e.g. Can attach a message to be revealed if Test positive

Can use IBE to shorten keys. Ciphertext still too long.
Predicate Encryption
Predicate Encryption

Comparison predicates (given Enc(a), for \( a \in [1, n] \), check if \( a \geq q \))
Predicate Encryption

- Comparison predicates (given $\text{Enc}(a)$, for $a \in [1,n]$, check if $a \geq q$)
- Can use a “set-hiding” broadcast encryption for intervals
Predicate Encryption

- Comparison predicates (given $\text{Enc}(a)$, for $a \in [1,n]$, check if $a \geq q$)
- Can use a “set-hiding” broadcast encryption for intervals
- Will see in next lecture
Predicate Encryption

- Comparison predicates (given $\text{Enc}(a)$, for $a \in [1,n]$, check if $a \geq q$)
- Can use a “set-hiding” broadcast encryption for intervals
  - Will see in next lecture
    - Idea: create ciphertexts that can be decrypted by keys in a range. To encrypt $a$, encrypt a random message addressed to the range $[a,n]$. Test key is the key for index $q$. 

Predicate Encryption

- Comparison predicates (given $\text{Enc}(a)$, for $a \in [1,n]$, check if $a \geq q$)
- Can use a "set-hiding" broadcast encryption for intervals
  - Will see in next lecture
    - Idea: create ciphertexts that can be decrypted by keys in a range. To encrypt $a$, encrypt a random message addressed to the range $[a,n]$. Test key is the key for index $q$.
  - Extends to range checking
Conjunctive Predicates
Conjunctive Predicates

Predicates of the form \((\phi_1(a_1) \text{ AND } \ldots \text{ AND } \phi_n(a_m))\)
Conjunctive Predicates

Predicates of the form \((\phi_1(a_1) \text{ AND } \ldots \text{ AND } \phi_n(a_m))\)

Should not reveal which clauses were not satisfied, if any
Conjunctive Predicates

Predicates of the form \((\phi_1(a_1) \text{ AND } \ldots \text{ AND } \phi_n(a_m))\)

- Should not reveal which clauses were not satisfied, if any
- e.g. in [BW07] \(\phi_i\) can be equality check \((a=w?)\), comparison \((a \geq q?)\), range check \((a \in [r,s]?)\) or membership in a list \((a \in S?)\)
Conjunctive Predicates

- Predicates of the form \((\phi_1(a_1) \, \text{AND} \ldots \text{AND} \, \phi_n(a_m))\)

- Should not reveal which clauses were not satisfied, if any

- e.g. in [BW07] \(\phi_i\) can be equality check \((a=w?)\), comparison \((a \geq q?)\), range check \((a \in [r,s]?)\) or membership in a list \((a \in S?)\)

- Tool: Hidden Vector matching, in which each \(\phi_i\) is an equality check or a don't care
Conjunctive Predicates

Predicates of the form \((\phi_1(a_1) \text{ AND } \ldots \text{ AND } \phi_n(a_m))\)

- Should not reveal which clauses were not satisfied, if any

- e.g. in [BW07] \(\phi_i\) can be equality check \((a=w?)\), comparison \((a \geq q?)\), range check \((a \in [r,s]?)\) or membership in a list \((a \in S?)\)

- Tool: Hidden Vector matching, in which each \(\phi_i\) is an equality check or a don’t care

- e.g. Using hidden vector matching to implement a conjunctive comparison predicate: for all \(i\), \(a_i \geq r_i\)
Predicates of the form \((\varphi_1(a_1) \text{ AND } ... \text{ AND } \varphi_n(a_m))\)

Should not reveal which clauses were not satisfied, if any

e.g. in [BW07] \(\varphi_i\) can be equality check \((a=w?)\), comparison \((a \geq q?)\), range check \((a \in [r,s]?)\) or membership in a list \((a \in S?)\)

Tool: Hidden Vector matching, in which each \(\varphi_i\) is an equality check or a don’t care

e.g. Using hidden vector matching to implement a conjunctive comparison predicate: for all \(i\), \(a_i \geq r_i\)

Check if binary \([X_{aij}]\) defined as \(X_{aij} = 1\) iff \(j \leq a_i\), matches with \([T_{rij}]\) defined as \(T_{rij} = 1\) if \(j \leq r_i\), and * otherwise
Conjunctive Predicates
Conjunctive Predicates

Using hidden vector matching for set membership: $a \in S \subseteq [1,n]$?
Conjunctive Predicates

Using hidden vector matching for set membership: \( a \in S \subseteq [1,n] \)?

Set membership is a disjunction of equalities: can be represented as (the negation of) a conjunction of inequalities.
Conjunctive Predicates

Using hidden vector matching for set membership: \( a \in S \subseteq [1,n] \)?

Set membership is a disjunction of equalities: can be represented as (the negation of) a conjunction of inequalities.

Check if binary vector \( X_a \) defined as \( X_{ai} = 1 \) iff \( a = i \), matches with \( T_S \) defined as \( T_{Si} = 0 \) if \( i \notin S \), and * otherwise.
Conjunctive Predicates

Using hidden vector matching for set membership: \( a \in S \subseteq [1,n] \)?

- Set membership is a disjunction of equalities: can be represented as (the negation of) a conjunction of inequalities

- Check if binary vector \( X^a \) defined as \( X^a_i = 1 \) iff \( a = i \), matches with \( T^S \) defined as \( T^S_i = 0 \) if \( i \not\in S \), and * otherwise

- Key and ciphertext proportional to size of universe \([1,n]\)
Conjunctive Predicates

- Using hidden vector matching for set membership: $a \in S \subseteq [1,n]$?

- Set membership is a disjunction of equalities: can be represented as (the negation of) a conjunction of inequalities

- Check if binary vector $X^a$ defined as $X^a_i = 1$ iff $a=i$, matches with $T^S$ defined as $T^S_i = 0$ if $i \notin S$, and * otherwise

- Key and ciphertext proportional to size of universe $[1,n]$

- Can extend to conjunction with other predicates
Conjunctive Predicates

- Using hidden vector matching for set membership: $a \in S \subseteq [1,n]$?

- Set membership is a disjunction of equalities: can be represented as (the negation of) a conjunction of inequalities

- Check if binary vector $X^a$ defined as $X^a_i = 1$ iff $a=i$, matches with $T^S$ defined as $T^S_i = 0$ if $i \notin S$, and * otherwise

- Key and ciphertext proportional to size of universe $[1,n]$

- Can extend to conjunction with other predicates

- More efficient set membership?
Bloom Filters
Bloom Filters

Elements $x$ in the universe mapped to $n$-bit binary vectors $h(x)$
Bloom Filters

- Elements $x$ in the universe mapped to $n$-bit binary vectors $h(x)$
- A subset $S$ is represented by $H(S) = \bigvee_{x \in S} h(x)$ (i.e., bit-wise OR)
Bloom Filters

- Elements $x$ in the universe mapped to $n$-bit binary vectors $h(x)$
- A subset $S$ is represented by $H(S) = \lor_{x \in S} h(x)$ (i.e., bit-wise OR)
- Given $H(S)$, to check if $x \in S$, for each coordinate $i$ s.t. $h(x)_i = 1$, check that $H(S)_i = 1$
Bloom Filters

- Elements $x$ in the universe mapped to $n$-bit binary vectors $h(x)$
- A subset $S$ is represented by $H(S) = \bigvee_{x \in S} h(x)$ (i.e., bit-wise OR)
- Given $H(S)$, to check if $x \in S$, for each coordinate $i$ s.t $h(x)_i = 1$, check that $H(S)_i = 1$
- No false negatives
Bloom Filters

- Elements $x$ in the universe mapped to $n$-bit binary vectors $h(x)$
- A subset $S$ is represented by $H(S) = \bigvee_{x \in S} h(x)$ (i.e., bit-wise OR)
- Given $H(S)$, to check if $x \in S$, for each coordinate $i$ s.t. $h(x)_i = 1$, check that $H(S)_i = 1$

- No false negatives

- False positive if all $i$ s.t. $h(x)_i = 1$ are covered by $h(x')$ for a set of other values $x'$
Bloom Filters

- Elements $x$ in the universe mapped to $n$-bit binary vectors $h(x)$
- A subset $S$ is represented by $H(S) = \bigvee_{x \in S} h(x)$ (i.e., bit-wise OR)
- Given $H(S)$, to check if $x \in S$, for each coordinate $i$ s.t $h(x)_i = 1$, check that $H(S)_i = 1$
- No false negatives
- False positive if all $i$ s.t. $h(x)_i = 1$ are covered by $h(x')$ for a set of other values $x'$
  - If $h$ is a random function with outputs of weight $d$, can bound the false positive rate in terms of $n$, $d$ and $|S|$
**Bloom Filters**

- Elements $x$ in the universe mapped to $n$-bit binary vectors $h(x)$
- A subset $S$ is represented by $H(S) = \bigvee_{x \in S} h(x)$ (i.e., bit-wise OR)
- Given $H(S)$, to check if $x \in S$, for each coordinate $i$ s.t $h(x)_i = 1$, check that $H(S)_i = 1$
  
- No false negatives
  
- False positive if all $i$ s.t. $h(x)_i = 1$ are covered by $h(x')$ for a set of other values $x'$
  
- If $h$ is a random function with outputs of weight $d$, can bound the false positive rate in terms of $n$, $d$ and $|S|$
  
- Or $h$ a CRHF with range being indices of a “cover free set system”
Set-Membership Predicate with Bloom Filters
Set-Membership Predicate with Bloom Filters

To check $a \in S \subseteq U$, where the universe $U$ can be large
Set-Membership Predicate with Bloom Filters

To check $a \in S \subseteq U$, where the universe $U$ can be large

Checking if $a \in S$ amounts to checking if the vector $h(a)$ is covered by $H(S)$
Set-Membership Predicate with Bloom Filters

- To check $a \in S \subseteq U$, where the universe $U$ can be large

- Checking if $a \in S$ amounts to checking if the vector $h(a)$ is covered by $H(S)$

- Implemented using hidden vector matching
Set-Membership Predicate with Bloom Filters

To check $a \in S \subseteq U$, where the universe $U$ can be large

Checking if $a \in S$ amounts to checking if the vector $h(a)$ is covered by $H(S)$

Implemented using hidden vector matching

$S$ encrypted: $T^a$ defined as: $T^a_i = 1$ if $h(a)_i = 1$, else *
Set-Membership Predicate with Bloom Filters

- To check \( a \in S \subseteq U \), where the universe \( U \) can be large.

- Checking if \( a \in S \) amounts to checking if the vector \( h(a) \) is covered by \( H(S) \).

- Implemented using hidden vector matching.

- \( S \) encrypted: \( T^a \) defined as: \( T^a_i = 1 \) if \( h(a)_i = 1 \), else *

- \( a \) encrypted: \( T^S \) defined as: \( T^S_i = 0 \) if \( H(S) = 0 \), else *
Inner-product Predicate
Attribute \( a \) is a vector. Predicate \( P_v \) is also specified by a vector \( v \): \( P_v(a) = 1 \) iff \( \langle v,a \rangle = 0 \)
Inner-product Predicate

 Attribute $a$ is a vector. Predicate $P_v$ is also specified by a vector $v$: $P_v(a) = 1$ iff $\langle v, a \rangle = 0$

 Or function $P_v$: $P_v(a,m) = m$ iff $\langle v, a \rangle = 0$, else $\perp$
Inner-product Predicate

- Attribute $a$ is a vector. Predicate $P_v$ is also specified by a vector $v$: $P_v(a) = 1$ iff $<v,a> = 0$

- Or function $P_v$: $P_v(a,m)=m$ iff $<v,a>=0$, else $\perp$

- General enough to capture several applications
Inner-product Predicate

Attribute \( a \) is a vector. Predicate \( P_v \) is also specified by a vector \( v \): \( P_v(a) = 1 \) iff \( \langle v, a \rangle = 0 \)

Or function \( P_v: P_v(a,m) = m \) iff \( \langle v, a \rangle = 0 \), else \( \bot \)

General enough to capture several applications

- e.g. Anonymous IBE from Inner-Product PE (with attached messages) over attributes in \( \mathbb{Z}_N \times \mathbb{Z}_N \)
Inner-product Predicate

- Attribute $a$ is a vector. Predicate $P_v$ is also specified by a vector $v$: $P_v(a) = 1$ iff $\langle v, a \rangle = 0$

- Or function $P_v$: $P_v(a,m)=m$ iff $\langle v, a \rangle=0$, else $\perp$

- General enough to capture several applications

- E.g. Anonymous IBE from Inner-Product PE (with attached messages) over attributes in $\mathbb{Z}_N \times \mathbb{Z}_N$

- For encrypting to identity id use attribute $a_{id} = (1, id)$. $SK_{id}$ is the test key for predicate with $v_{id} = (-id, 1)$. Anonymity: attribute remains hidden if no matching $SK$ given.
Inner-product Predicate
Inner-product Predicate

- Can be used to get Hidden Vector matching predicate
Inner-product Predicate

- Can be used to get Hidden Vector matching predicate
- Map a given pattern vector of length m to a vector v in $(\mathbb{Z}_N)^{2m}$ by mapping * to (0,0) and a to (1,a).
Inner-product Predicate

- Can be used to get Hidden Vector matching predicate

- Map a given pattern vector of length m to a vector v in \((\mathbb{Z}_N)^{2m}\) by mapping * to (0,0) and a to (1,a).

- Map the hidden attribute vector u to a vector a by mapping each co-ordinate \(u_i\) to \((-r_i.u_i, r_i)\), for random \(r_i\)
Inner-product Predicate

- Can be used to get Hidden Vector matching predicate

- Map a given pattern vector of length m to a vector v in $(\mathbb{Z}_N)^{2m}$ by mapping * to (0,0) and a to (1,a).

- Map the hidden attribute vector $u$ to a vector $a$ by mapping each co-ordinate $u_i$ to $(-r_i \cdot u_i, r_i)$, for random $r_i$

- If pattern matches $u$, then $\langle v, a \rangle = 0$
Inner-product Predicate

- Can be used to get Hidden Vector matching predicate

- Map a given pattern vector of length $m$ to a vector $v$ in $(\mathbb{Z}_N)^{2m}$ by mapping * to (0,0) and a to (1,a).

- Map the hidden attribute vector $u$ to a vector $a$ by mapping each co-ordinate $u_i$ to $(-r_i u_i, r_i)$, for random $r_i$

- If pattern matches $u$, then $<v,a>=0$

- Random $r_i$ to avoid cancelations while summing, so that if pattern does not match, w.h.p $<v,a>\neq 0$
**Inner-product Predicate**

- Can be used to get Hidden Vector matching predicate

- Map a given pattern vector of length m to a vector v in \((\mathbb{Z}_N)^{2m}\) by mapping * to (0,0) and a to (1,a).

- Map the hidden attribute vector u to a vector a by mapping each co-ordinate \(u_i\) to \((-r_i.u_i, r_i)\), for random \(r_i\)

- If pattern matches u, then \(\langle v, a \rangle = 0\)

- Random \(r_i\) to avoid cancelations while summing, so that if pattern does not match, w.h.p \(\langle v, a \rangle \neq 0\)

- Can support * in both the pattern and the hidden vector
Inner-product Predicate
Inner-product Predicate

Other predicates implied:
Inner-product Predicate

Other predicates implied:

- Polynomials: \( P_v \) can be a polynomial (represented as a vector of co-efficients) and attribute a the value (represented as the vector \( <1,a,a^2,\ldots,a^d> \)) at which \( P_v \) is evaluated, or vice versa.
Inner-product Predicate

Other predicates implied:

Polynomials: \( P_v \) can be a polynomial (represented as a vector of co-efficients) and attribute a the value (represented as the vector \( <1,a,a^2,...,a^d> \)) at which \( P_v \) is evaluated, or vice versa.

Disjunction \( (a_1=v_1) \) OR \( (a_2=v_2) \): polynomial \( (a_1-v_1) \) \( (a_2-v_2) \)
Inner-product Predicate

Other predicates implied:

Polynomials: $P_v$ can be a polynomial (represented as a vector of co-efficients) and attribute a the value (represented as the vector $<1,a,a^2,...,a^d>$) at which $P_v$ is evaluated, or vice versa.

- Disjunction $(a_1=v_1) \text{ OR } (a_2=v_2)$: polynomial $(a_1-v_1)(a_2-v_2)$
- Conjunction $(a_1=v_1) \text{ AND } (a_2=v_2)$: $r_1(a_1-v_1) + r_2(a_2-v_2)$
Inner-product Predicate

Other predicates implied:

Polynomials: $P_v$ can be a polynomial (represented as a vector of co-efficients) and attribute a the value (represented as the vector $<1,a,a^2,...,a^d>$) at which $P_v$ is evaluated, or vice versa

- Disjunction $(a_1=v_1) \lor (a_2=v_2)$: polynomial $(a_1-v_1)(a_2-v_2)$

- Conjunction $(a_1=v_1) \land (a_2=v_2)$: $r_1(a_1-v_1) + r_2(a_2-v_2)$

Exact threshold: for $A, V \subseteq [1,n]$, $P_{V,t}(A) = 1$ iff $|A \cap V|=t$
Inner-product Predicate

Other predicates implied:

- Polynomials: $P_v$ can be a polynomial (represented as a vector of co-efficients) and attribute a the value (represented as the vector $<1,a,a^2,\ldots,a^d>$) at which $P_v$ is evaluated, or vice versa.

- Disjunction $(a_1=v_1) \text{ OR } (a_2=v_2)$: polynomial $(a_1-v_1) (a_2-v_2)$

- Conjunction $(a_1=v_1) \text{ AND } (a_2=v_2)$: $r_1(a_1-v_1) + r_2(a_2-v_2)$

- Exact threshold: for $A, V \subseteq [1,n]$, $P_{V,t}(A) = 1 \iff |A \cap V|=t$

- Map $V$ to $v$ as $v_0=1$ and for $i=1$ to $n$, $v_i = 1 \iff i \in V$. Map $A$ to a vector $a$ where $a_0 = -t$, for $i=1$ to $n$, $a_i = 1 \iff i \in A$. 
Predicate/Functional Encryption
Predicate/Functional Encryption

- Constructions using bilinear pairings known [KSW08,LOSTW10,OT10]
Predicate/Functional Encryption

- Constructions using bilinear pairings known [KSW08,LOSTW10,OT10]
- Supports inner product predicates (and more)
Predicate/Functional Encryption

- Constructions using bilinear pairings known [KSW08,LOSTW10,OT10]
- Supports inner product predicates (and more)
- Can base security on Decision Linear assumption
Predicate/Functional Encryption

- Constructions using bilinear pairings known \([\text{KSW08,LOSTW10,OT10}]\)
- Supports inner product predicates (and more)
- Can base security on Decision Linear assumption
- Can get CCA security
Today
Today

🔍 Searching on Encrypted Data
Today

- Searching on Encrypted Data
  - To check if encrypted keyword matches a given keyword
Today

- Searching on Encrypted Data
  - To check if encrypted keyword matches a given keyword
  - From anonymous IBE
Today

- Searching on Encrypted Data
  - To check if encrypted keyword matches a given keyword
  - From anonymous IBE
- Predicate/Functional encryption
Today

- Searching on Encrypted Data
  - To check if encrypted keyword matches a given keyword
  - From anonymous IBE
- Predicate/Functional encryption
  - To check if encrypted attributes satisfy a given predicate
Today

- Searching on Encrypted Data
  - To check if encrypted keyword matches a given keyword
  - From anonymous IBE

- Predicate/Functional encryption
  - To check if encrypted attributes satisfy a given predicate
  - Hidden vector matching, inner-product predicate, ...