Searching on/Testing Encrypted Data

Lecture 20

Searchable Encryption

Searchable Encryption

- A test key T_w that allows one to test if $Dec_{SK}(C) = w$

Searchable Encryption

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

Searchable Encryption

- A test key T_w that allows one to test if $Dec_{SK}(C) = w$
  - 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 $Dec_{SK}(C) = w$
  - 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 $Dec_{SK}(C) = w$
  - 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 $Dec_{SK}(C) = w$
  - 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)←KeyGen, T_w←TestKeyGen(SK,w), Enc_{PK}(w), Dec_{SK}(C) and $Test_{T_w}$(C)

Searchable Encryption

- Components: (PK,SK)←KeyGen, T_w←TestKeyGen(SK,w), $Enc_{PK}(w)$, $Dec_{SK}(C)$ and $Test_{T_w}(C)$

- Correctness: For all (possibly adversarially chosen) words w, for C←$Enc_{PK}(w)$, we have $Dec_{SK}(C)$ = w and $Test_{T_w}(C)$=1. For any other (adversarially chosen) word w', $Test_{T_{w'}}(C)$=0.

Searchable Encryption

- Components: $(PK,SK) \leftarrow KeyGen$, $T_w \leftarrow TestKeyGen(SK,w)$, $Enc_{PK}(w)$, $Dec_{SK}(C)$ and $Test_{T_w}(C)$
- Correctness: For all (possibly adversarially chosen) words w, for $C \leftarrow Enc_{PK}(w)$, we have $Dec_{SK}(C) = w$ and $Test_{T_w}(C)=1$. For any other (adversarially chosen) word w', $Test_{T_{w'}}(C)=0$.
    - 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)←KeyGen, T_w←TestKeyGen(SK,w), $Enc_{PK}(w)$, $Dec_{SK}(C)$ and $Test_{Tw}(C)$
- Correctness: For all (possibly adversarially chosen) words w, for C←$Enc_{PK}(w)$, we have $Dec_{SK}(C)$ = w and $Test_{Tw}(C)$=1. For any other (adversarially chosen) word w', $Test_{Tw'}(C)$=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).
- Secrecy: CPA or CCA security against adversary with oracle access to 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) | w in dictionary}

Trivial Solution: using PKE

- If the dictionary is small, (PK,SK) = { (PK_w,SK_w) | w in dictionary}
- To encrypt a keyword (or, in fact, a list of keywords), $Enc_{PK}(w)$= <$Enc_{PK1}(0)$, ..., $Enc_{PKw}(1)$, ..., $Enc_{PKn}(0)$>

Trivial Solution: using PKE

- If the dictionary is small, (PK,SK) = { (PK_w,SK_w) | w in dictionary}
- To encrypt a keyword (or, in fact, a list of keywords), $Enc_{PK}(w)$= <$Enc_{PK1}(0)$, ..., $Enc_{PKw}(1)$, ..., $Enc_{PKn}(0)$>
  - TestKeyGen(SK,w) = SK_w

Trivial Solution: using PKE

- If the dictionary is small, (PK,SK) = { (PK_w,SK_w) | w in dictionary}
- To encrypt a keyword (or, in fact, a list of keywords), $Enc_{PK}(w)$= <$Enc_{PK1}(0)$, ..., $Enc_{PKw}(1)$, ..., $Enc_{PKn}(0)$>
  - 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 (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), $Enc_{PK}(w)$= <$IBEnc_{PK}(0;id=0)$, ..., $IBEnc_{PK}(1;id=w)$, ..., $IBEnc_{PK}(0;id=n)$>

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

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 $Enc_{PK}(w)= IBEnc_{PK}(1;id=w)$

PEKS from Anonymous IBE

- Suppose, to encrypt a keyword $Enc_{PK}(w) = IBEnc_{PK}(1; id=w)$
  - Secure?

PEKS from Anonymous IBE

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

PEKS from Anonymous IBE

- Suppose, to encrypt a keyword $Enc_{PK}(w) = IBEnc_{PK}(1; id=w)$
  - Secure?
- IBE ciphertexts may reveal id (can have the id in the clear)
- Anonymous IBE

PEKS from Anonymous IBE

- Suppose, to encrypt a keyword $Enc_{PK}(w)= IBEnc_{PK}(1;id=w)$
  - 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

PEKS from Anonymous IBE

- Suppose, to encrypt a keyword $Enc_{PK}(w) = IBEnc_{PK}(1;id=w)$
  - 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 $Enc_{PK}(w)= IBEnc_{PK}(1;id=w)$
  - 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)
- 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, $Enc_{PK}(w) = (IBEnc_{PK}(r;id=w),r)$ 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, $Enc_{PK}(w)$= ($IBEnc_{PK}$(r;id=w),r) for a random message r (|r|=k)
  - If decrypting $IBEnc_{PK}$(r;id=w), for a <u>random</u> 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, $Enc_{PK}(w)$= ($IBEnc_{PK}$(r;id=w),r) for a random message r (|r|=k)

  - If decrypting $IBEnc_{PK}$(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, $Enc_{PK}(w)$= ($IBEnc_{PK}(r;id=w)$,r) for a random message r (|r|=k)
  - If decrypting $IBEnc_{PK}(r;id=w)$, 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
  - Or add such "decryption recognition" directly to Anonymous IBE

Predicate Encryption

Predicate Encryption

- Test for properties of encryped attributes

Predicate Encryption

- Test for properties of encryped attributes
  - For $C \leftarrow Enc_{PK}(a)$, we require that boolean $Test_{T_P}(C)=1$ iff $P(a)=1$

Predicate Encryption

- Test for properties of encryped attributes
  - For $C \leftarrow Enc_{PK}(a)$, we require that boolean $Test_{T_P}(C)=1$ iff $P(a)=1$
    - Or $Test_{T_P}(C) = P(a)$, for a function P (e.g. $P(a,m)=m$ iff $P'(a)=1$)

Predicate Encryption

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

Predicate Encryption

- Test for properties of encryped attributes
  - For $C \leftarrow Enc_{PK}(a)$, we require that boolean $Test_{T_P}(C)=1$ iff $P(a)=1$
    - Or $Test_{T_P}(C) = P(a)$, for a function P (e.g. $P(a,m)=m$ iff $P'(a)=1$)
- 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 ($a \in S$?)

Predicate Encryption

- Test for properties of encryped attributes
  - For $C \leftarrow Enc_{PK}(a)$, we require that boolean $Test_{T_P}(C)=1$ iff $P(a)=1$
    - Or $Test_{T_P}(C) = P(a)$, for a function P (e.g. $P(a,m)=m$ iff $P'(a)=1$)
- 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

Predicate Encryption

- Test for properties of encryped attributes
  - For $C \leftarrow Enc_{PK}(a)$, we require that boolean $Test_{T_P}(C)=1$ iff $P(a)=1$
    - Or $Test_{T_P}(C) = P(a)$, for a function P (e.g. $P(a,m)=m$ iff $P'(a)=1$)
- 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
  - $(PK,SK)=\{(PK_P,SK_P) \mid P$ in the predicate family$\}$. Ciphertext has $Enc_{PK_P}(P(a))$ for each P.

Predicate Encryption

- Test for properties of encryped attributes
  - For $C \leftarrow Enc_{PK}(a)$, we require that boolean $Test_{T_P}(C)=1$ iff $P(a)=1$
    - Or $Test_{T_P}(C) = P(a)$, for a function P (e.g. $P(a,m)=m$ iff $P'(a)=1$)
- 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 $Enc_{PK_P}(P(a))$ for each P.
    - Can support functions instead of predicates

Predicate Encryption

- Test for properties of encryped attributes
  - For $C \leftarrow Enc_{PK}(a)$, we require that boolean $Test_{T_P}(C)=1$ iff $P(a)=1$
    - Or $Test_{T_P}(C) = P(a)$, for a function P (e.g. $P(a,m)=m$ iff $P'(a)=1$)
- 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
  - $(PK,SK)=\{(PK_P,SK_P) \mid P$ in the predicate family$\}$. Ciphertext has $Enc_{PK_P}(P(a))$ 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 encryped attributes
  - For $C \leftarrow Enc_{PK}(a)$, we require that boolean $Test_{T_P}(C)=1$ iff $P(a)=1$
    - Or $Test_{T_P}(C) = P(a)$, for a function P (e.g. $P(a,m)=m$ iff $P'(a)=1$)
- 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 $Enc_{PK_P}(P(a))$ 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 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 Enc(a), for $a \in [1,n]$, check if $a \geq q$)
- Can use a "set-hiding" broadcast encryption for intervals
  - Given $a \in [1,n]$, $Enc_{PK}(a)$ uses the broadcast encryption to encrypt a random nonce r for the interval [a,n]

Predicate Encryption

- Comparison predicates (given Enc(a), for $a \in [1,n]$, check if $a \geq q$)
- Can use a "set-hiding" broadcast encryption for intervals
  - Given $a \in [1,n]$, $Enc_{PK}(a)$ uses the broadcast encryption to encrypt a random nonce r for the interval [a,n]
    - To test if $a \geq q$, T_q is the decryption key for "user" q. Test positive iff decrypting gives r. (Alternatively, incorporate decryption recognition into the BE.)

Predicate Encryption

- Comparison predicates (given Enc(a), for $a \in [1,n]$, check if $a \geq q$)
- Can use a "set-hiding" broadcast encryption for intervals
  - Given $a \in [1,n]$, $Enc_{PK}(a)$ uses the broadcast encryption to encrypt a random nonce r for the interval [a,n]
    - To test if $a \geq q$, T_q is the decryption key for "user" q. Test positive iff decrypting gives r. (Alternatively, incorporate decryption recognition into the BE.)
  - Set-hiding BE for intervals with $O(\sqrt{n})$ long ciphertexts using bilinear pairings known

Predicate Encryption

- Comparison predicates (given Enc(a), for $a \in [1,n]$, check if $a \geq q$)
- Can use a "set-hiding" broadcast encryption for intervals
  - Given $a \in [1,n]$, $Enc_{PK}(a)$ uses the broadcast encryption to encrypt a random nonce r for the interval [a,n]
    - To test if $a \geq q$, T_q is the decryption key for "user" q. Test positive iff decrypting gives r. (Alternatively, incorporate decryption recognition into the BE.)
  - Set-hiding BE for intervals with $O(\sqrt{n})$ long ciphertexts using bilinear pairings known
  - Extends to range checking

Conjunctive Predicates

Conjunctive Predicates

- Predicates of the form ($\phi_1(a_1)$ AND .... AND $\phi_n(a_m)$)

Conjunctive Predicates

- Predicates of the form ($\phi_1(a_1)$ AND .... AND $\phi_n(a_m)$)
  - Should not reveal which clauses were not satisfied, if any

Conjunctive Predicates

- Predicates of the form ($\phi_1(a_1)$ AND .... AND $\phi_n(a_m)$)
  - Should not reveal which clauses were not satisfied, if any
  - e.g. in [BW07] ϕ_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 ($\phi_1(a_1)$ AND .... AND $\phi_n(a_m)$)
  - Should not reveal which clauses were not satisfied, if any
  - e.g. in [BW07] ϕ_i can be equality check ($a=w$?), comparison ($a \geq q$?), range check ($a \in [r,s]$?) or membership in a list ($a \in S$?)
    - Using Hidden Vector matching: each ϕ_i is an equality check or a don't care

Conjunctive Predicates

- Predicates of the form ($\phi_1(a_1)$ AND .... AND $\phi_n(a_m)$)
  - Should not reveal which clauses were not satisfied, if any
  - e.g. in [BW07] ϕ_i can be equality check ($a=w$?), comparison ($a \geq q$?), range check ($a \in [r,s]$?) or membership in a list ($a \in S$?)
    - Using Hidden Vector matching: each ϕ_i is an equality check or a don't care
    - e.g. Using hidden vector matching to implement a <u>conjunctive comparison</u> predicate: for all i, $a_i \geq r_i$

Conjunctive Predicates

- Predicates of the form ($\phi_1(a_1)$ AND .... AND $\phi_n(a_m)$)
  - Should not reveal which clauses were not satisfied, if any
  - e.g. in [BW07] ϕ_i can be equality check ($a=w$?), comparison ($a \geq q$?), range check ($a \in [r,s]$?) or membership in a list ($a \in S$?)
    - Using Hidden Vector matching: each ϕ_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 $[X^a_{ij}]$ defined as $X^a_{ij} = 1$ iff $j \leq a_i$, matches with $[T^r_{ij}]$ defined as $T^r_{ij} = 1$ 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 $X^a_i = 1$ iff $a=i$, matches with T^S defined as $T^S_i = 0$ 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 $i \notin S$, 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 of set memberships

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 of set memberships
- 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 $B(S) = \vee_{x \in S} H(x)$ (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 $B(S) = \vee_{x \in S} H(x)$ (i.e., bit-wise OR)
- Given B(S), to check if $x \in S$, for each coordinate i s.t $H(x)_i = 1$, check that $B(S)_i = 1$

Bloom Filters

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

Bloom Filters

- Elements x in the universe mapped to n-bit binary vectors H(x)
- A subset S is represented by $B(S) = \vee_{x \in S} H(x)$ (i.e., bit-wise OR)
- Given B(S), to check if $x \in S$, for each coordinate i s.t $H(x)_i = 1$, check that $B(S)_i = 1$
  - 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 $B(S) = \vee_{x \in S} H(x)$ (i.e., bit-wise OR)
- Given B(S), to check if $x \in S$, for each coordinate i s.t $H(x)_i = 1$, check that $B(S)_i = 1$
  - 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 <u>random</u> function with outputs of <u>weight</u> 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 $B(S) = \vee_{x \in S} H(x)$ (i.e., bit-wise OR)
- Given B(S), to check if $x \in S$, for each coordinate i s.t $H(x)_i = 1$, check that $B(S)_i = 1$
  - 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 B(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 $B(S)$
    - 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 $B(S)$
    - Implemented using hidden vector matching
      - T^a defined as: $T^a_i = 1$ if $H(a)_i = 1$, else *

Inner-product Predicate

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 $\langle v,a \rangle = 0$
  - Or function P_v : $P_v(a,m)=m$ iff $\langle v,a \rangle=0$, else $\perp$
- General enough to capture several applications

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 using 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 $<v,a> = 0$
  - Or function P_v : $P_v(a,m)=m$ iff $<v,a>=0$, else $\perp$
- General enough to capture several applications
  - e.g. Anonymous IBE using Inner-Product PE (with attached messages) over attributes in $\mathbb{Z}_N \times \mathbb{Z}_N$
    - For encrypting to identity id use attribute (1,id). Predicate used as SK_{id} is (-id,1). Anonymity since attribute remains hidden if no matching SK

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 ($-r_i.u_i$, 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>≠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 $\langle 1,a,a^2,\ldots,a^d \rangle$) 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 $\langle 1,a,a^2,\ldots,a^d \rangle$) 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 $\langle 1,a,a^2,\ldots,a^d \rangle$) at which P_v is evaluated, or vice versa
    - Disjunction $(a_1=v_1)$ OR $(a_2=v_2)$: polynomial $(a_1-v_1)(a_2-v_2)$
    - Conjunction $(a_1=v_1)$ 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 $\langle 1,a,a^2,\ldots,a^d \rangle$) at which P_v is evaluated, or vice versa
    - Disjunction $(a_1=v_1)$ OR $(a_2=v_2)$: polynomial $(a_1-v_1)(a_2-v_2)$
    - Conjunction $(a_1=v_1)$ 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 $|A \cap V| = t$

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 $(a_1=v_1)$ OR $(a_2=v_2)$: polynomial $(a_1-v_1)\ (a_2-v_2)$
    - Conjunction $(a_1=v_1)$ 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 [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, ...