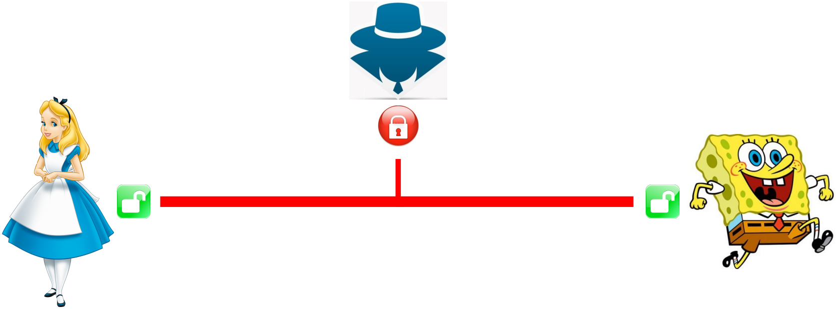
Pattern Matching in Encrypted Stream from Inner Product Encryption

Elie Bouscatié(1,2) Guilhem Castagnos(1) Olivier Sanders(2)

(1) Université de Bordeaux
(2) Orange Labs

PKC 2023, May 09, 2023

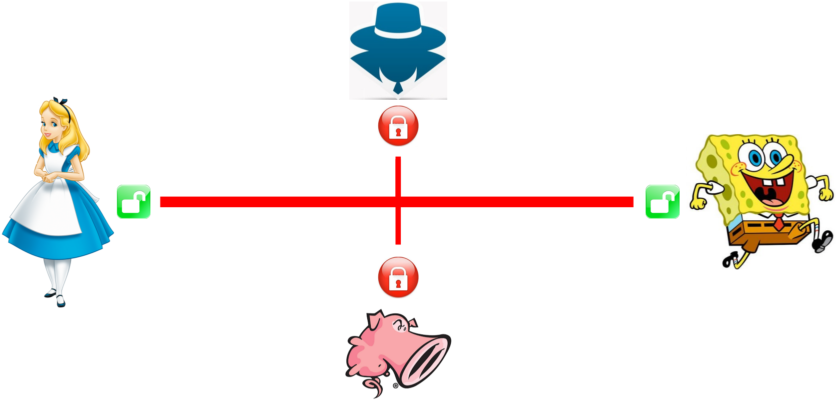
End-to-End Encryption



More and more encrypted data

- about 90% of Internet traffic worldwide is encrypted
- increased use of encrypted messaging services (WhatsApp,...)

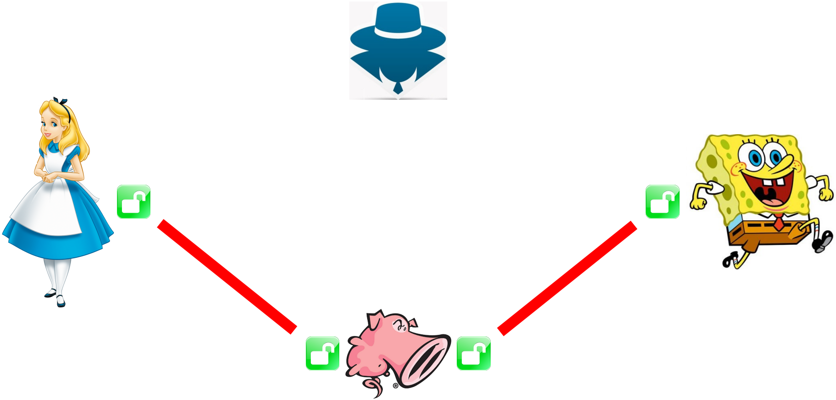
End-to-End Encryption



Standard encryption protocols designed to prevent any processing

- arbitration between privacy and functionalities

End-to-End Encryption



Current solutions imply decryption by a gateway

- the gateway can access all data exchanged through the channel
- what is the point of end-to-end encryption?

Pattern Matching

- Many applications perform pattern matching
  - Intrusion Detection System (IDS)
  - Content filtering
  - Searches on genomic data
  - ...
- Example of Snort rules:
  - alert tcp (msg:"MALWARE-BACKDOOR - Dagger_ 1.4.0"; content:"2| 00 00 00 06 00 00 00 | Drives | 24 00 |",depth 16;)
  - alert tcp (msg:"MALWARE-BACKDOOR QAZ Worm Client Login access"; content:"qazwsx.hsq";)

Searchable Encryption

- SE is an encryption scheme with additional features:
  - given sk one can derive a trapdoor td_w on a pattern W
  - given td_w, the gateway can test whether $C = \mathtt{Encrypt}(W)$

Searchable Encryption

- SE is an encryption scheme with additional features:
  - given sk one can derive a trapdoor td_w on a pattern W
  - given td_w, the gateway can test whether $C = \mathtt{Encrypt}(W)$

SE does not address our problem

Searchable Encryption

- SE is an encryption scheme with additional features:
  - given sk one can derive a trapdoor td_w on a pattern W
  - given td_w, the gateway can test whether $C = \texttt{Encrypt}(W)$

SE does not address our problem

- We need:
  - given sk one can derive a trapdoor td_w on a pattern W
  - given td_w, the gateway can check whether W is contained in the stream encrypted in C

Dealing with Data Streams

SE-based solutions follow the sliding window method:

- Each C_i can be tested using td_W
- The process must be repeated for each possible length of keywords

Dealing with Data Streams

SE-based solutions follow the sliding window method:

$ h o s t i l e

E_K

C_0 C_1

patterns

host

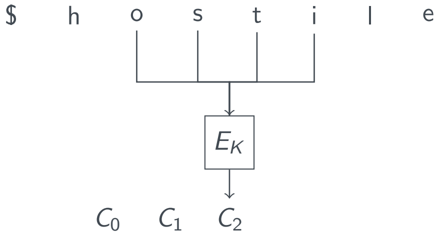
hostile

...

- Each C_i can be tested using td_W
- The process must be repeated for each possible length of keywords

Dealing with Data Streams

SE-based solutions follow the sliding window method:



patterns

host

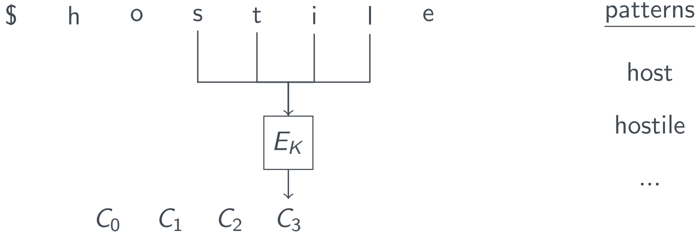
hostile

...

- Each C_i can be tested using td_W
- The process must be repeated for each possible length of keywords

Dealing with Data Streams

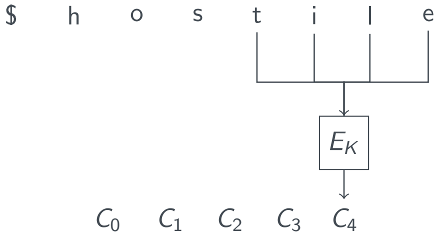
SE-based solutions follow the sliding window method:



- Each C_i can be tested using td_W
- The process must be repeated for each possible length of keywords

Dealing with Data Streams

SE-based solutions follow the sliding window method:



$ h o s t i l e

E_K

C_0 C_1 C_2 C_3 C_4

patterns

host

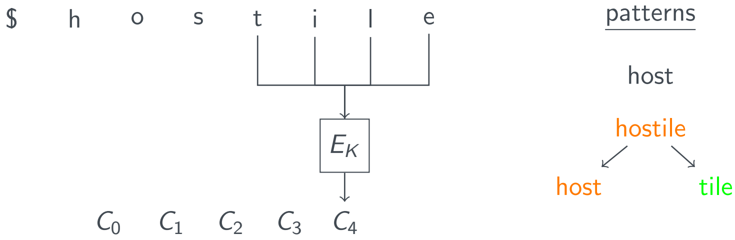
hostile

...

- Each C_i can be tested using td_W
- The process must be repeated for each possible length of keywords

Dealing with Data Streams

SE-based solutions follow the sliding window method:



- Each C_i can be tested using td_W
- Splitting keywords harms privacy

Hidden Vector Encryption

- n-HVE enables to encrypt vectors $\mathbf{x} = (x_1, \ldots, x_n) \in \Sigma^n$
- Secret keys are associated with vectors $\mathbf{k} = (k_1, \ldots, k_n)$ where
  - $k_i \in \Sigma$
  - or $k_i = *$ (wildcard)
- Given $\mathsf{sk}_{\mathbf{k}}$ and an encryption of $\mathbf{x}$, `Test` returns
  - 1 if $\forall i \in [1, n]$ $k_i = *$ or $k_i = x_i$
  - 0 otherwise
- No leakage beyond the output of $\texttt{Test}(\mathsf{sk}_{\mathbf{k}}, \mathbf{x})$

Dealing with Data Streams

plaintext	\$	h	o	s	t	i	l	e
$\mathbf{k}_{\texttt{host},0}$	h	o	s	t	∗	∗	∗	∗
$\mathbf{k}_{\texttt{host},1}$	∗	h	o	s	t	∗	∗	∗
$\mathbf{k}_{\texttt{host},2}$	∗	∗	h	o	s	t	∗	∗
$\mathbf{k}_{\texttt{host},3}$	∗	∗	∗	h	o	s	t	∗
$\mathbf{k}_{\texttt{host},4}$	∗	∗	∗	∗	h	o	s	t

pattern: host

- Wildcards enable to deal with offsets
- Secret keys must be issued for each possible offset

$\Rightarrow$ n keys per pattern for n bytes stream

SEPM

- Searchable Encryption supporting Pattern Matching (SEPM) is essentially a HVE with key size independent of n
- State-of-the-Art:

DFOS18 [1]: First construction, $O(n)$ public key, proof in the GGM

BCC20 [2]: public key size independent of n, proof in the GGM

BCS21 [3]: improved performance, selective security under static assumption

Ad-hoc constructions, no adaptive security under standard assumptions

---

[1] Desmoulins *et al.* Pattern Matching on Encrypted Streams. Asiacrypt 2018.

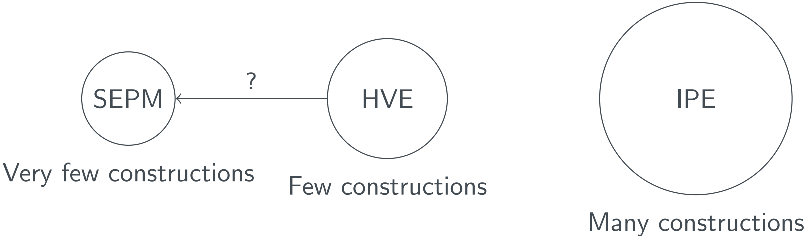
[2] Bkakria *et al.* Privacy-Preserving Pattern Matching on Encrypted Data. Asiacrypt 2020.

[3] Bouscatié *et al.* Public Key Encryption with Flexible Pattern Matching. Asiacrypt 2021.

Our Contribution

Our Goal

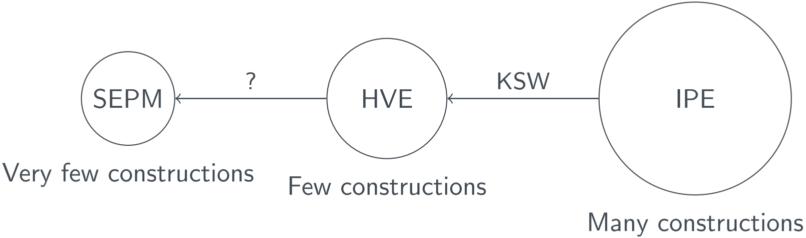
We want to identify/improve relations between primitives



- No known relation between SEPM and HVE

Our Goal

We want to identify/improve relations between primitives



- No known relation between SEPM and HVE
- Generic transformation (KSW[4]) from IPE to HVE with two-fold ciphertext increase

---

[4] Katz *et al.* Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. Journal of Cryptology 2013.

First Step: HVE → SEPM

[BCC20] introduced the fragmentation approach:

- streams are split into overlapping fragments
- d is a bound on the largest searchable pattern

$$\mathbf{x} = \rlap{$\overbrace{\phantom{x_1, \dots, x_d, x_{d+1}, \dots, x_{2d}}}^{\mathbf{x}_1}$} x_1, \dots, x_d, \rlap{$\underbrace{\phantom{x_{d+1}, \dots, x_{2d}, x_{2d+1}, \dots, x_{3d}}}_{\mathbf{x}_2}$} x_{d+1}, \dots, x_{2d}, \rlap{$\overbrace{\phantom{x_{2d+1}, \dots, x_{3d}, x_{3d+1}, \dots, x_{4d}}}^{\mathbf{x}_3}$} x_{2d+1}, \dots, x_{3d}, \rlap{$\underbrace{\phantom{x_{3d+1}, \dots, x_{4d}, x_{4d+1}, \dots, x_{5d}}}_{\mathbf{x}_4}$} x_{3d+1}, \dots, x_{4d}, x_{4d+1}, \dots, x_{5d}, \dots$$

Fragmentation is much more powerful than initially thought

First Step: HVE → SEPM

We use fragmentation to circumvent HVE limitations for streams

- streams are split into overlapping fragments
- d is a bound on the largest searchable pattern

$$\mathbf{x} = \overbrace{x_1,\ldots,x_d,\mathrlap{\underbrace{\phantom{x_{d+1},\ldots,x_{2d},x_{2d+1},\ldots,x_{3d}}}_{\mathbf{x}_2}}x_{d+1},\ldots,x_{2d}}^{\mathbf{x}_1},\overbrace{x_{2d+1},\ldots,x_{3d},\mathrlap{\underbrace{\phantom{x_{3d+1},\ldots,x_{4d},x_{4d+1},\ldots,x_{5d}}}_{\mathbf{x}_4}}x_{3d+1},\ldots,x_{4d}}^{\mathbf{x}_3},x_{4d+1},\ldots,x_{5d},\ldots$$

- Use $2d$-HVE to encrypt each fragment
- Use wildcards to deal with pattern offsets within a fragment

O(d) key size (independent of n)

- Run HVE.`Test` in each relevant fragment

Second Step: IPE $\rightarrow$ HVE (KSW)

KSW proposed a generic conversion $2n$-IPE $->$ n-HVE

$$\mathbf{x'} = (\quad x_1 \cdot r_1 \quad , \quad \ldots \quad , \quad x_n \cdot r_n \quad , \quad -r_1 \quad , \quad \ldots \quad , \quad -r_n \quad)$$

- HVE.$\mathtt{Encrypt}(x_1, \ldots, x_n)$: $r \overset{\$}{\leftarrow} \mathbb{Z}_p$ and $C \leftarrow$ IPE.$\mathtt{Encrypt}(\mathbf{x'})$

Second Step: IPE $\rightarrow$ HVE (KSW)

KSW proposed a generic conversion $2n$-IPE $->$ n-HVE

$$\begin{array}{ll} \mathbf{x'} = (& x_1 \cdot r_1, \; \ldots, \; x_n \cdot r_n, \; -r_1, \; \ldots, \; -r_n \;) \\ \mathbf{k'} = (& 1, \; \ldots, \; 1, \; k_1, \; \ldots, \; k_n \;) \end{array}$$

- HVE.`Keygen`$(k_1, \ldots, k_n)$: $\mathsf{sk}_{\mathbf{k}} \leftarrow$ IPE.`Keygen`$(\mathbf{k'})$

Second Step: IPE $\rightarrow$ HVE (KSW)

KSW proposed a generic conversion $2n$-IPE $->$ n-HVE

$$\begin{array}{ll} \mathbf{x'} = (& x_1 \cdot r_1 \quad , \quad \ldots \quad , \quad x_n \cdot r_n \quad , \quad -r_1 \quad , \quad \ldots \quad , \quad -r_n \quad) \\ \mathbf{k'} = (& 1 \quad , \quad \ldots \quad , \quad 1 \quad , \quad k_1 \quad , \quad \ldots \quad , \quad k_n \quad) \end{array}$$

- HVE.`Test`$(C, \mathsf{sk}_{\mathbf{k}})$:return 1 iff $< \mathbf{x'}, \mathbf{k'} >= 0$ using IPE.`Eval`$(C, \mathsf{sk}_{\mathbf{k}})$

Correctness:

- $(x_1, \ldots, x_n) = (k_1, \ldots, k_n) \Rightarrow < \mathbf{x'}, \mathbf{k'} >= 0$
- $< \mathbf{x'}, \mathbf{k'} >= 0 \Rightarrow (x_1, \ldots, x_n) = (k_1, \ldots, k_n)$ with overwhelming probability...

Second Step: IPE → HVE (KSW)

KSW proposed a generic conversion $2n$-IPE $->$ n-HVE

$$\begin{array}{llllllll} \mathbf{x'}=(& x_1 \cdot r_1 & , \ldots , & x_n \cdot r_n & , & -r_1 & , \ldots , & -r_n \;) \\ \mathbf{k'}=(& 1 & , \ldots , & 1 & , & k_1 & , \ldots , & k_n \;) \end{array}$$

- HVE.`Test`$(C, \mathsf{sk}_{\mathbf{k}})$:return 1 iff $< \mathbf{x'}, \mathbf{k'} >= 0$ using IPE.`Eval`$(C, \mathsf{sk}_{\mathbf{k}})$

Correctness:

- $(x_1, \ldots, x_n) = (k_1, \ldots, k_n) \Rightarrow < \mathbf{x'}, \mathbf{k'} >= 0$
- $< \mathbf{x'}, \mathbf{k'} >= 0 \Rightarrow (x_1, \ldots, x_n) = (k_1, \ldots, k_n)$ with overwhelming probability...

for honest key queries!

Second Step: IPE $\rightarrow$ HVE (KSW)

KSW proposed a generic conversion $2n$-IPE $->$ n-HVE

$$\begin{array}{lccccccc} \mathbf{x'}=(& x_1 \cdot r_1 & , \ldots , & x_n \cdot r_n & , & -r_1 & , \ldots , & -r_n \;) \\ \mathbf{k'}=(& 1 & , \ldots , & 1 & , & k_1 & , \ldots , & k_n \;) \end{array}$$

- HVE.$\mathtt{Test}(C, \mathsf{sk}_{\mathbf{k}})$:return 1 iff $< \mathbf{x'}, \mathbf{k'} >= 0$ using IPE.$\mathtt{Eval}(C, \mathsf{sk}_{\mathbf{k}})$

Security:

- no formal security proof in KSW
- malicious queries create discrepancy between IPE/HVE experiments:
  - Given the challenge IPE.$\mathtt{Encrypt}(\mathbf{x'})$, $\mathcal{A}$ chooses $\mathbf{k'}$ such that $< \mathbf{x'}, \mathbf{k'} >= 0$ but $(x_1, \ldots, x_n) \neq (k_1, \ldots, k_n)$
  - Valid query for the HVE experiment but not for the IPE one
- We show how to circumvent this issue in our paper

Second Step: IPE $\rightarrow$ HVE (Ours)

We introduce a generic conversion $(n+1)$-IPE $->$ n-HVE

$$\begin{array}{lccccccc}
\mathbf{x'}=(& x_1 & , & \ldots & , & x_n & , & \text{-1}\ ) \\
\mathbf{k'}=(& r_1 & , & \ldots & , & r_n & , & <\mathbf{k},\mathbf{r}>\ )
\end{array}$$

- two-fold improvement compared to KSW
- $(x_1, \ldots, x_n) = (k_1, \ldots, k_n) \Rightarrow < \mathbf{x'}, \mathbf{k'} >= 0$
- $< \mathbf{x'}, \mathbf{k'} >= 0 \Rightarrow (x_1, \ldots, x_n) = (k_1, \ldots, k_n)$ with overwhelming probability... still for honest key queries!

Second Step: IPE $\rightarrow$ HVE (Ours)

We introduce a generic conversion $(n+1)$-IPE $->$ n-HVE

$$\begin{array}{lcccccccc} \mathbf{x'}=(& x_1 & , & \ldots & , & x_n & , & \text{-1} &) \\ \mathbf{k'}=(& r_1 & , & \ldots & , & r_n & , & <\mathbf{k},\mathbf{r}> &) \end{array}$$

- two-fold improvement compared to KSW
- $(x_1,\ldots,x_n)=(k_1,\ldots,k_n) \Rightarrow <\mathbf{x'},\mathbf{k'}>=0$
- $<\mathbf{x'},\mathbf{k'}>=0 \Rightarrow (x_1,\ldots,x_n)=(k_1,\ldots,k_n)$ with overwhelming probability... still for honest key queries!

Security

- Selective security holds without additional assumptions
- Adaptive security needs an additional assumption

Second Step: IPE → HVE (Ours)

Adaptive security experiment

$$\mathcal{C}_{IPE} \qquad\qquad\qquad\qquad \mathcal{C}_{HVE} \qquad\qquad\qquad\qquad \mathcal{A}_{HVE}$$

$$\mathcal{C}_{HVE} \xleftarrow{\mathbf{k}_i} \mathcal{A}_{HVE}$$

$$\mathcal{C}_{IPE} \xleftarrow{\mathbf{k'}_i = (r_1, \ldots, r_n, < \mathbf{k}, \mathbf{r} >)} \mathcal{C}_{HVE} \qquad \mathbf{r} \overset{\$}{\leftarrow} \mathbb{Z}_p$$

$$\mathcal{C}_{IPE} \xrightarrow{\mathsf{sk}_{\mathbf{k'}_i}} \mathcal{C}_{HVE} \xrightarrow{\mathsf{sk}_{\mathbf{k'}_i}} \mathcal{A}_{HVE}$$

Second Step: IPE → HVE (Ours)

Adaptive security experiment

$\mathcal{C}_{IPE}$ $\mathcal{C}_{HVE}$ $\mathcal{A}_{HVE}$

$$\xleftarrow{\mathbf{k}_i}$$

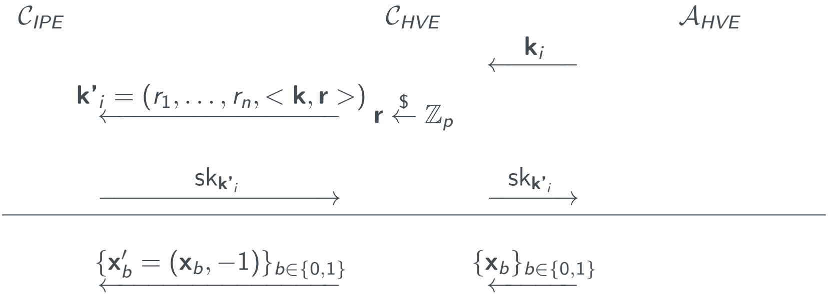
$$\xleftarrow{\mathbf{k'}_i = (r_1, \ldots, r_n, <\mathbf{k}, \mathbf{r}>)} \quad \mathbf{r} \overset{\$}{\leftarrow} \mathbb{Z}_p$$

$$\xrightarrow{\mathsf{sk}_{\mathbf{k'}_i}} \qquad \xrightarrow{\mathsf{sk}_{\mathbf{k'}_i}}$$

$$\xleftarrow{\{\mathbf{x}'_b = (\mathbf{x}_b, -1)\}_{b \in \{0,1\}}} \qquad \xleftarrow{\{\mathbf{x}_b\}_{b \in \{0,1\}}}$$ (Challenge phase)

Second Step: IPE → HVE (Ours)

Adaptive security experiment



$\mathcal{C}_{IPE}$ $\mathcal{C}_{HVE}$ $\mathcal{A}_{HVE}$

$\xleftarrow{\mathbf{k}_i}$

$\xleftarrow{\mathbf{k'}_i = (r_1, \ldots, r_n, < \mathbf{k}, \mathbf{r} >)}$ $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_p$

$\xrightarrow{\mathsf{sk}_{\mathbf{k'}_i}}$ $\xrightarrow{\mathsf{sk}_{\mathbf{k'}_i}}$

$\xleftarrow{\{\mathbf{x}'_b = (\mathbf{x}_b, -1)\}_{b \in \{0,1\}}}$ $\xleftarrow{\{\mathbf{x}_b\}_{b \in \{0,1\}}}$

Example of malicious selection of $\mathbf{x}_0$ and $\mathbf{x}_1$

- $\forall i, b, \mathbf{k}_i \neq \mathbf{x}_b$ (valid HVE queries)
- $\exists i^*, b^* :< \mathbf{k}'_{i*}, \mathbf{x}_{b*} >= 0$ and $< \mathbf{k}'_{i^*}, \mathbf{x}_{1-b^*} > \neq 0$ (invalid IPE queries)

Second Step: IPE $\rightarrow$ HVE (Ours)

Main observations

- Knowledge of $\mathbf{k}'_i$ enables to find such $\mathbf{x}_b$

  $\Rightarrow$ $\mathsf{sk}_{\mathbf{k}'_i}$ must hide $\mathbf{k}'_i$

- Function-privacy strongly depends on the context (pattern entropy)

  $\Rightarrow$ we want a property independent of the context

- $\mathcal{A}$ has some control over $\mathbf{k}'_i$ as it selects $\mathbf{k}_i$

  $\Rightarrow$ must be modelled by our property

Second Step: IPE $\rightarrow$ HVE (Ours)

We introduce key privacy for IPE

$\mathcal{C}_{IPE}$ $\qquad\qquad\qquad\qquad$ $\mathcal{A}(\mathsf{pk})$

$\xleftarrow{\quad\mathbf{k}_i\quad}$

$\xrightarrow{\quad\mathsf{sk}_{\mathbf{k}_i}\quad}$

$\xleftarrow{\text{(challenge) } \mathbf{y}}$

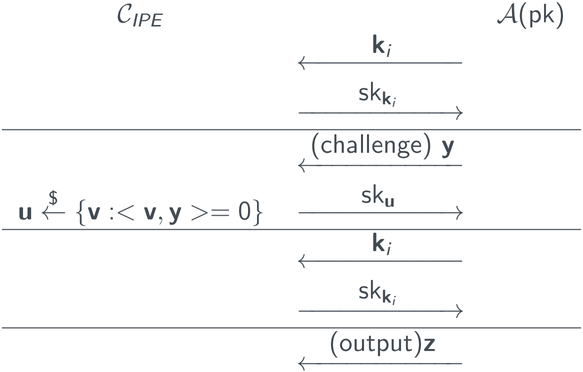
$\mathbf{u} \overset{\$}{\leftarrow} \{\mathbf{v} :< \mathbf{v}, \mathbf{y} >= 0\}$ $\quad \xrightarrow{\quad\mathsf{sk}_{\mathbf{u}}\quad}$

$\mathcal{A}$ has some control on $\mathbf{u}$ through the choice of $\mathbf{y}$

sufficient for our IPE→HVE transformation

Second Step: IPE → HVE (Ours)

We introduce key privacy for IPE



$\mathcal{C}_{IPE}$ $\mathcal{A}(\mathsf{pk})$

$\xleftarrow{\mathbf{k}_i}$

$\xrightarrow{\mathsf{sk}_{\mathbf{k}_i}}$

(challenge) $\xleftarrow{\mathbf{y}}$

$\mathbf{u} \overset{\$}{\leftarrow} \{\mathbf{v} :< \mathbf{v}, \mathbf{y} >= 0\}$ $\xrightarrow{\mathsf{sk}_{\mathbf{u}}}$

$\xleftarrow{\mathbf{k}_i}$

$\xrightarrow{\mathsf{sk}_{\mathbf{k}_i}}$

(output) $\xleftarrow{\mathbf{z}}$

$\mathcal{A}$ wins if:

- $< \mathbf{u}, \mathbf{z} >= 0$
- $\mathbf{z} \notin \mathbf{Vect}(\mathbf{y})$ (no trivial win)

Second Step: IPE → HVE (Ours)

- We prove adaptive security of our conversion with key private IPE
- Key privacy can easily be assessed
  - independent of the context
  - achieved by KSW under Discrete Log assumption
  - achieved by OT[5] under Discrete Log assumption
  - we provide examples of scheme that do not achieve it

⇒ some IPE schemes yield better HVE than others

---

[5] Okamoto *et al.* Adaptively attribute-hiding inner product encryption. Eurocrypt 2012.

Conclusion

- We show how to generically build SEPM from HVE
- We revisit relations between HVE and IPE
  - We introduce a more efficient conversion IPE → HVE than KSW
  - We prove selective security of our conversion without additional condition
  - We prove adaptive security assuming a mild condition we formalise
- Our work leads to many new SEPM constructions with new features