VSSE: application to FOIA

Monica Pardeshi, Shengyu Ge and Milind Kumar V

Overview

- FOIA
  - What
  - Simplification
  - SSE
    - Threat model
- Verifiable SSE
  - Algorithm
  - Security properties
  - System demonstration
- Extensions

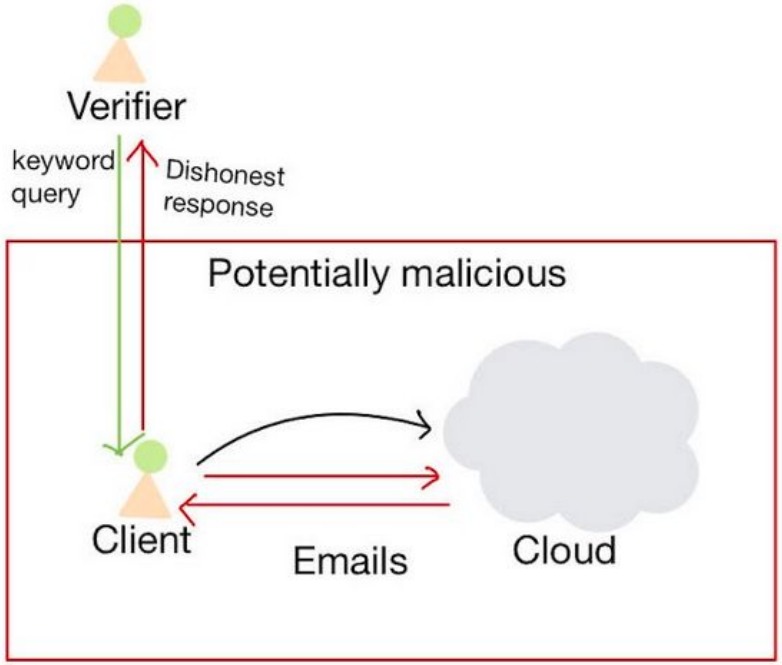
The FOIA setting

FOIA

- Request information from government/University
  - Emails
  - Files
  - Documentation
  - Voicemail messages
- Examples
  - Collaboration between industry and academia
  - The Amazon Ring case (2019, [1])
    - Privacy concerns
    - Data handling

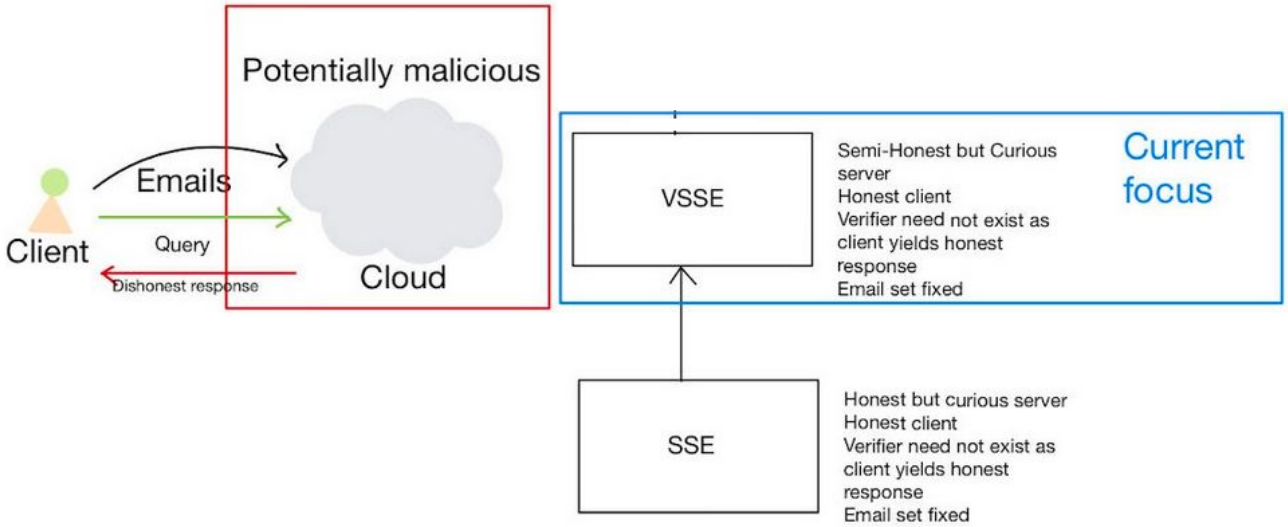
FOIA- emails by keyword

- Given keywords, return all containing emails
- Desired
  - Preserve privacy of email owner
    - Avoid storing plaintext on cloud
  - Prevent manipulation by server
  - Prevent manipulation by client



SSE

- Symmetric Searchable Encryption
- Honest client



VSSE

Threat model

- Adversarial CSP
  - Will store provided emails without tampering or deletion
  - May try to learn information about underlying plaintext
  - May suppress a fraction of the emails from a search query
- IND-CKA2 security
- UF-CKA security

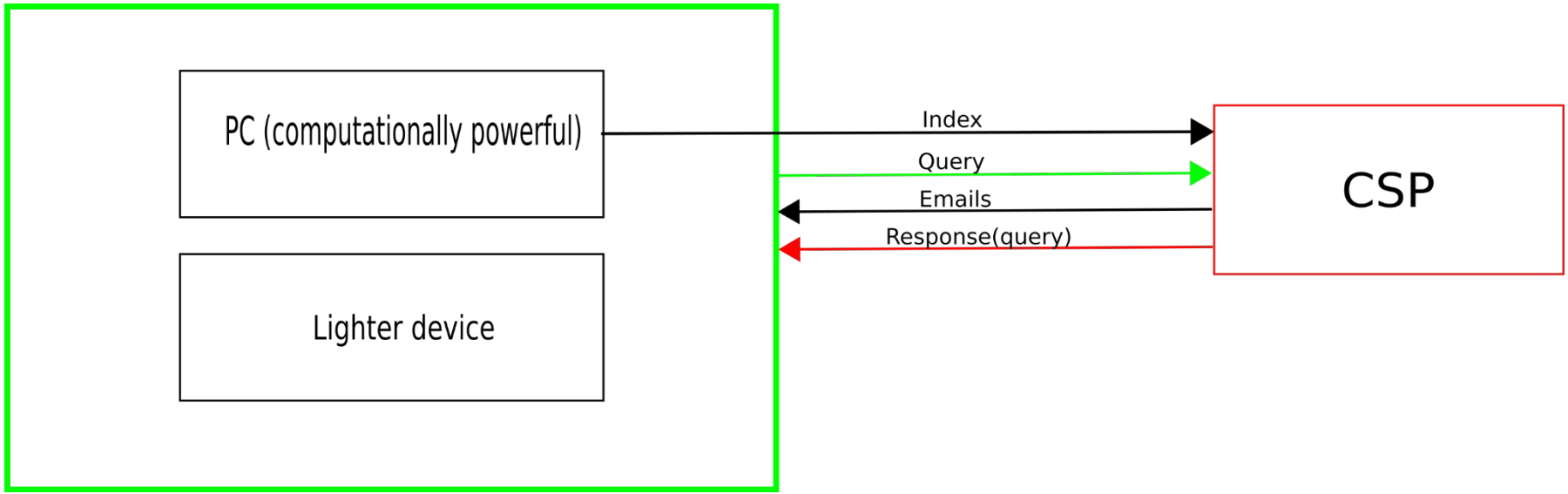
Algorithm

- $\langle \mathbf{K}, Params \rangle \leftarrow KeyGen(\kappa)$: *is a probabilistic algorithm that takes the security parameters* κ *as input and outputs the key* $\mathbf{K}$ *and system parameters.*
- $\mathcal{I} \leftarrow BuildIndex(\mathbf{F}, \mathbf{K})$: *is a probabilistic algorithm executed by the user, that takes the set of plaintext files* $\mathbf{F}$ *and the key* $\mathbf{K}$ *as input and outputs the secure index* $\mathcal{I}$.
- $\tau_s \leftarrow SearchToken(w, \mathbf{K})$: *is a deterministic algorithm executed by the user, that takes the key* $\mathbf{K}$ *and search keyword* w *as input and outputs the search token* τ_s.

Algorithm

- $\langle x, Tag \rangle \leftarrow Search(\tau_s, \mathcal{I})$: *is a deterministic algorithm run by the CSP, which takes the secure index* $\mathcal{I}$ *and search token* τ_s *as input and outputs the obscured bitmap* x *and verification tag* Tag.
- $\{True, False\} \leftarrow Verify(Tag, x, id(w))$: *is an algorithm executed by the user, that takes verification tag* Tag, *obscured bitmap* x, *keyword identifier* $id(w)$ *and key* $\mathbf{K}$ *as input and outputs verification result of search outcome.*

System model



Implementation Details

KeyGen(κ) : This algorithm is run by the user to generate the set of keys used in the scheme. Choose three cryptographic MAC's defined as follows:

- $H_1 : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$
- $H_2 : \{0,1\}^\kappa \times \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}^\kappa$
- $H_3 : \{0,1\}^\kappa \times \{0,1\}^n \times \{0,1\}^\kappa \to \{0,1\}^\kappa$

Where, the first inputs are cryptographic keys. Output the key $\mathbf{K} = \langle K_1, K_2, K_e, K_h \rangle \xleftarrow{R} \{0,1\}^\kappa$

Implementation Details

BuildIndex$(\mathbf{F}, \mathbf{K})$: This algorithm is run by the user to generate and output the secure index $\mathcal{I} = \langle T_f, T_s \rangle$. **Generation of T_f:** For $i = 1$ to m, compute $c_i = SKE.Enc(K_e, f_i)$ and store the tuple $\langle i, c_i \rangle$ as a row in T_f.

Generation of T_s: Extract the keywords in $\mathbf{F}$ and set $\mathbf{W} = \{w_0, \ldots, w_n\}$ and for all w_i in $\mathbf{W}$:

- Generate identifier $id(w_i) = H_1(K_1, w_i)$
- Choose $r_i \xleftarrow{R} \{0,1\}^\kappa$ and obtain mask $h_i = H_2(K_2, w_i, r_i)$
- Create the bitmap $B_m(w_i)[j] = 1$ for all $j \in$ index of encrypted files having the keyword w_i.
- Compute $x_i = B_m(w_i) \oplus h_i$ and $Tag_i = H_3(K_h, B_m(w_i), id(w_i))$
- Store $\langle key, value, Tag \rangle = \langle id(w_i), x_i || r_i, Tag_i \rangle$ as a row in T_s.

Implementation Details

Verify$(\mathbf{x_i}||\mathbf{r_i}, \mathbf{Tag_i})$: This algorithm is run by the user to verify the correctness of the search outcome sent by CSP.

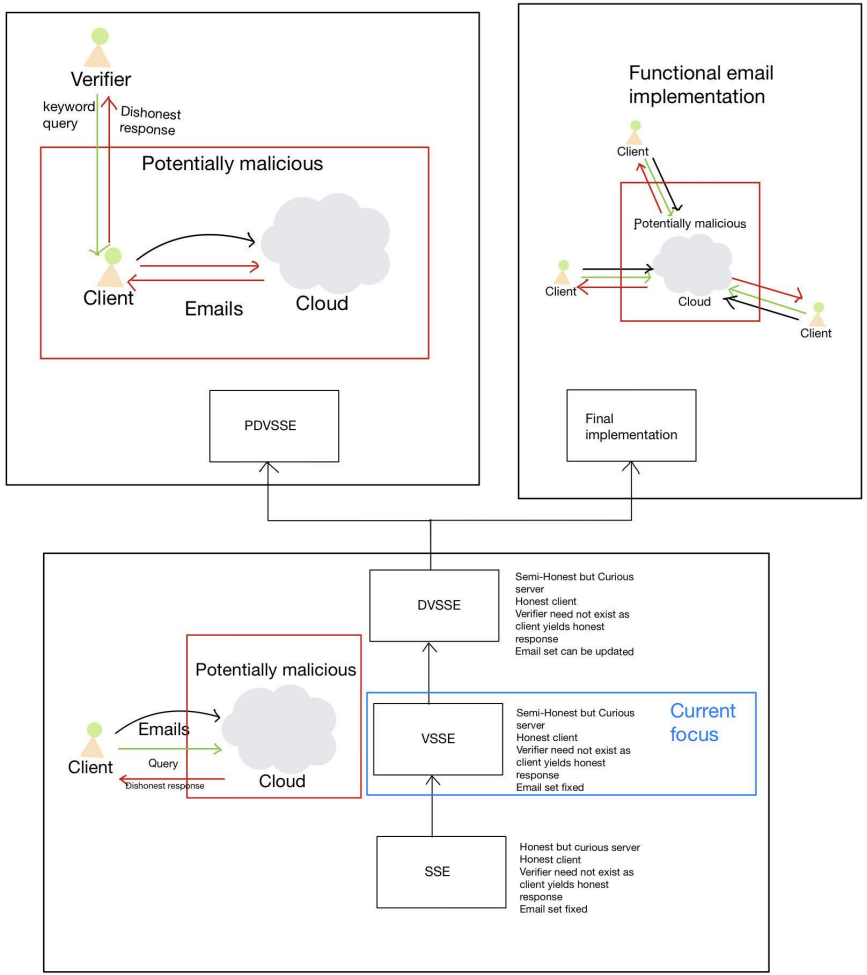
- Parse and extract r_i from $x_i||r_i$.
- Compute the keyword identifier $id(w_i) = H_1(K_1, w_i)$.
- Compute the mask $h_i = H_2(K_2, w_i, r_i)$.
- Extract the bitmap $B_m(w_i) = x_i \oplus h_i$.
- Calculate $Tag'_i = H_3(K_h, B_m(w_i), id(w_i))$.
- If $(Tag'_i = Tag_i)$ output $True$; else output $False$.

Implementation Details

- Use HMAC
  - A secure PRF is a secure MAC
  - HMAC is a secure PRF as long as the compression function is a secure PRF (2014, [2])
- Use AES-CFB
  - CPA secure
  - No padding required

Demo

Extensions


Verifier
keyword query
Dishonest response
Potentially malicious
Client
Emails
Cloud
PDVSSE
Functional email implementation
Client
Potentially malicious
Client
Cloud
Client
Final implementation
DVSSE
Semi-Honest but Curious server
Honest client
Verifier need not exist as client yields honest response
Email set can be updated
Potentially malicious
Emails
Client
Query
Dishonest response
Cloud
VSSE
Semi-Honest but Curious server
Honest client
Verifier need not exist as client yields honest response
Email set fixed
Current focus
SSE
Honest but curious server
Honest client
Verifier need not exist as client yields honest response
Email set fixed

References

---

1. https://sgandlur.com/category/amazon-ring/
2. https://eprint.iacr.org/2006/043.pdf

Thank you!