Private Stream Search at the Same Communication Cost as a Regular Search

Matthieu Finiasz and Kannan Ramchandran

- A stream search consists in filtering data according to a set of keywords:
  - the data is a stream (it could also be a database)
    - → every piece of data is treated independently
  - the filtering is done externally
    - → you retrieve only the matching data.
- A typical scenario is Google Alerts:
  - get an alert for each new page matching your interests.
- Private Stream Search does this without revealing the keywords (your interests) to the filtering server.

- Protect your privacy:
  - Google Alerts,
  - web search in general...
- Protect your financial interests:
  - when searching for patents,
    $\longrightarrow$ reveals what your company is focusing on.
- Global surveillance systems:
  - search for keywords in emails.
- But PSS is only worth it if it is efficient:
  - no one is ready to lose efficiency for privacy...

- To preserve privacy, the user sends a masked query:
  - a public list of possible keywords is needed,
  - the query is an encrypted selection of keywords.
- The server filters according to the encrypted query:
  - all documents/all keywords are treated symmetrically,
  - it accumulates matches in an encrypted data buffer,
  - only the user can extract the matches.
- PSS requires computations on encrypted data:
  - this is possible using homomorphic encryption.

- Paillier's encryption function $\mathcal{E}$ has the properties:

$$\mathcal{E}(m + m') = \mathcal{E}(m)\mathcal{E}(m'),$$
$$\mathcal{E}(c \times m) = \mathcal{E}(m)^c.$$

  $\longrightarrow$ possible to perform operations on encrypted values.

- A fully homomorphic scheme would also allow to compute $\mathcal{E}(m \times m')$ from $\mathcal{E}(m)$ and $\mathcal{E}(m')$.

- Paillier's cryptosystem is semantically secure:
  - impossible to distinguish $\mathcal{E}(0)$ and $\mathcal{E}(1)$.

The First PSS Scheme

[Ostrovsky-Skeith 2005]

- Requirements for this scheme:
  - a public dictionary of keywords $\Omega = \{k_1, ..., k_{|\Omega|}\}$,
  - the users asks OR queries on words of Ω,
  - a database/stream of t documents $(f_1, ..., f_t)$,
  - the users has an estimate of the number m of matches.
- We consider an example with:

$\mathit{\Omega}$ = dog brown cat black bird white

f_1= "the dog is black" f_3= "the bird is white"

f_2= "the cat is white" f_4= "the bird is black"

$$\Omega = \text{dog} \quad \text{brown} \quad \boxed{\text{cat}} \quad \text{black} \quad \text{bird} \quad \boxed{\text{white}}$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1)$$

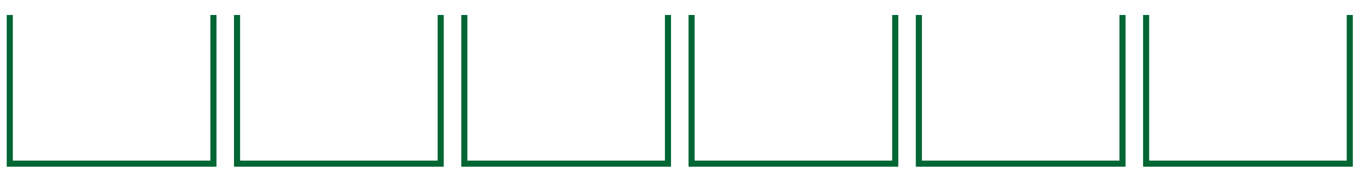
USER

- The user wants to query "cat OR white",
  - he computes a tuple Q of $\mathcal{E}(0)$ and $\mathcal{E}(1)$ accordingly.

$$\Omega = \text{dog} \quad \text{brown} \quad \text{cat} \quad \text{black} \quad \text{bird} \quad \text{white}$$

$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1)$$

SERVER



buffer B

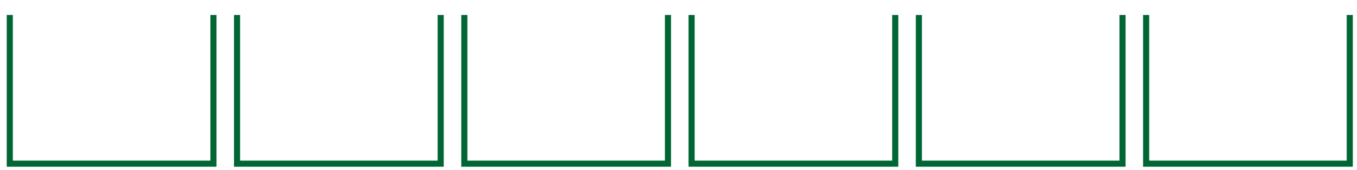
- The server prepares a response buffer B,
  - the matches will be accumulated in B.

$$\Omega = \text{dog} \quad \text{brown} \quad \text{cat} \quad \text{black} \quad \text{bird} \quad \text{white}$$

$$Q = \underline{\mathcal{E}(0)} \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \underline{\mathcal{E}(0)} \quad \mathcal{E}(0) \quad \mathcal{E}(1)$$

$f_1 =$ "the dog is black" $\qquad \mathcal{E}(0+0)$

SERVER



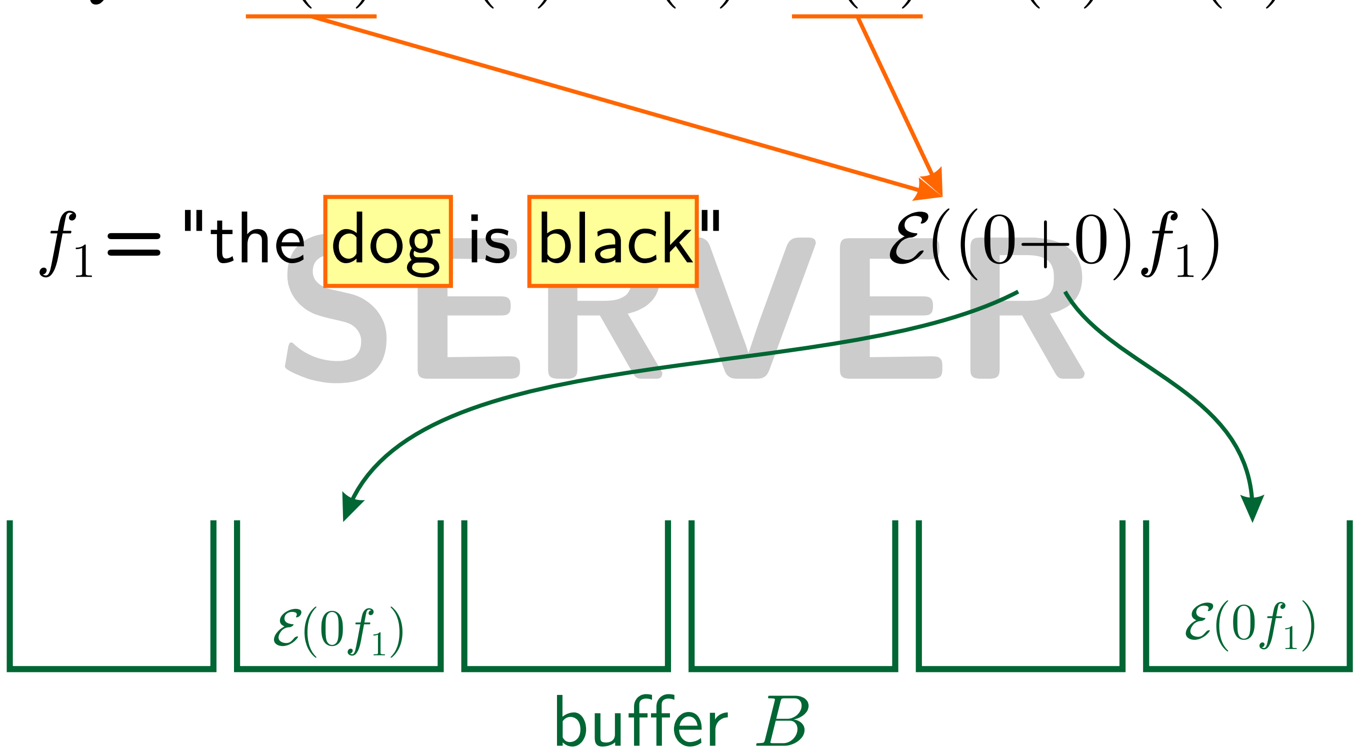
buffer B

- For every document f_i, the server computes:

$$\mathcal{E}(c_i) = \prod_{k_j \in f_i} q_j.$$

→ c_i is the number of matching keywords in f_i.

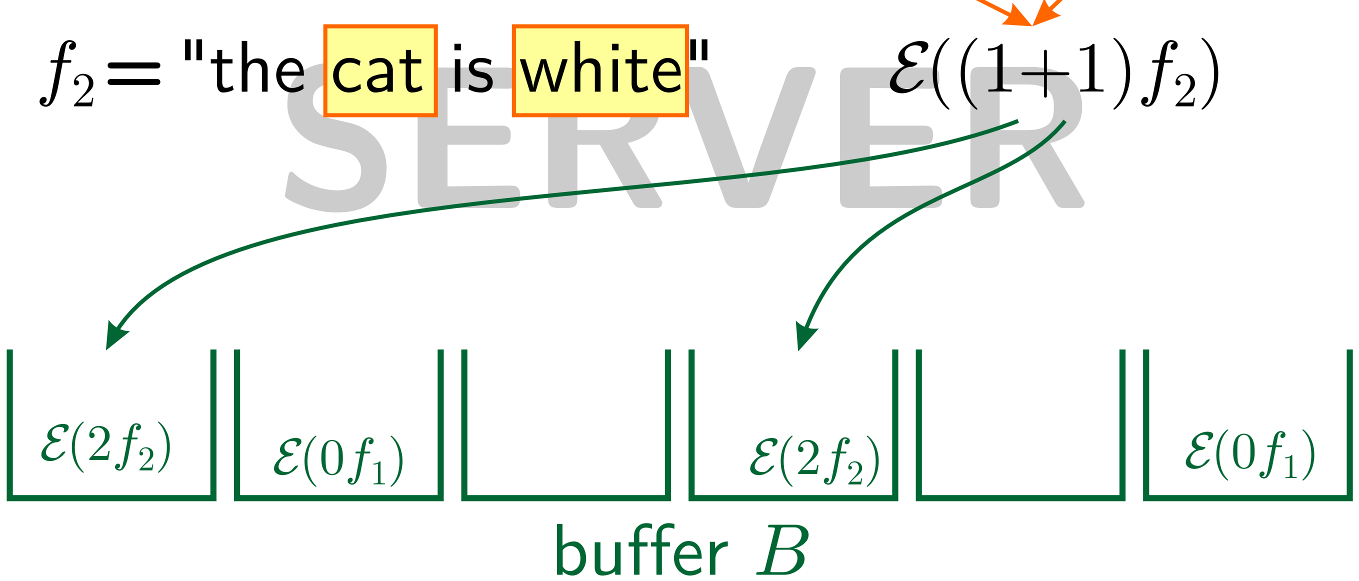
Ω	$=$	dog	brown	cat	black	bird	white
Q	$=$	$\mathcal{E}(0)$	$\mathcal{E}(0)$	$\mathcal{E}(1)$	$\mathcal{E}(0)$	$\mathcal{E}(0)$	$\mathcal{E}(1)$



- For every document f_i:
  - the server "adds" $\mathcal{E}(c_i)^{f_i} = \mathcal{E}(c_i f_i)$ randomly in B.

$$\Omega = \text{dog} \quad \text{brown} \quad \text{cat} \quad \text{black} \quad \text{bird} \quad \text{white}$$

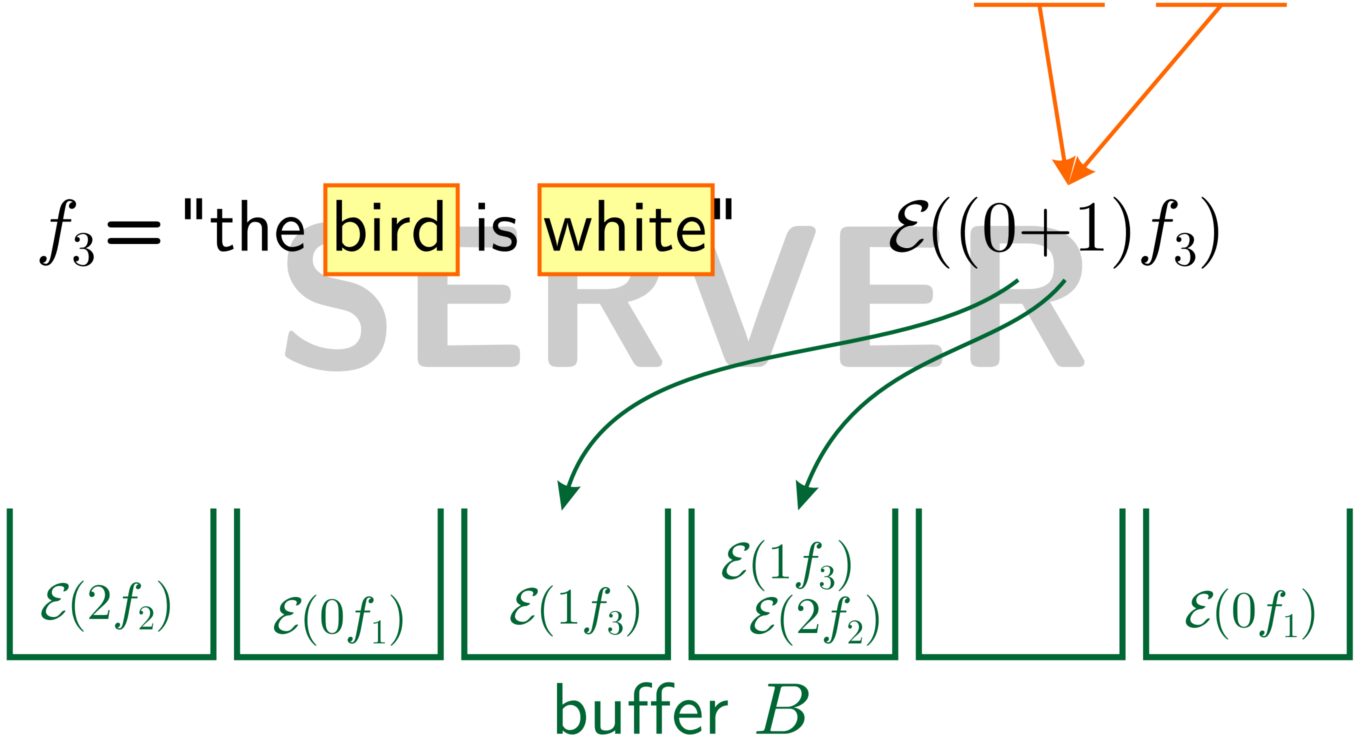
$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \underline{\mathcal{E}(1)} \quad \mathcal{E}(0) \quad \mathcal{E}(0) \quad \underline{\mathcal{E}(1)}$$



- The server repeats this for all documents.

$$\Omega = \text{dog} \quad \text{brown} \quad \text{cat} \quad \text{black} \quad \text{bird} \quad \text{white}$$

$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \mathcal{E}(0) \quad \underline{\mathcal{E}(0)} \quad \underline{\mathcal{E}(1)}$$

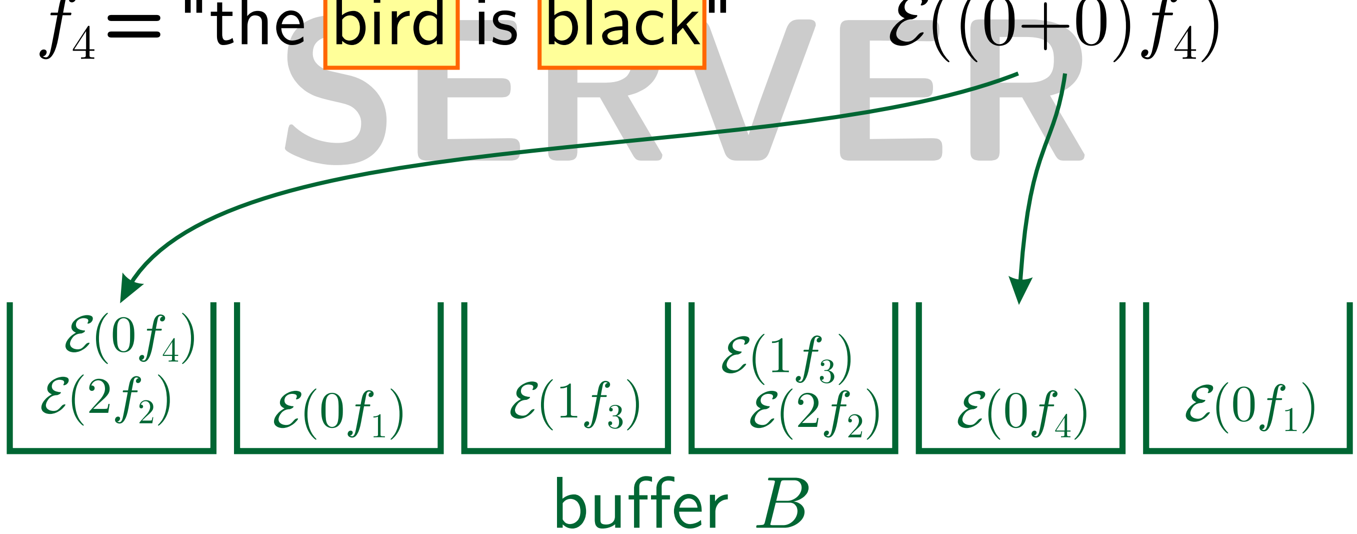


- The server repeats this for all documents.

$$\Omega = \text{dog} \quad \text{brown} \quad \text{cat} \quad \text{black} \quad \text{bird} \quad \text{white}$$

$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \underline{\mathcal{E}(0)} \quad \underline{\mathcal{E}(0)} \quad \mathcal{E}(1)$$

$f_4 =$ "the $\boxed{\text{bird}}$ is $\boxed{\text{black}}$" $\qquad \mathcal{E}((0+0)\,f_4)$



- The server repeats this for all documents.

$$\Omega = \text{dog} \quad \text{brown} \quad \text{cat} \quad \text{black} \quad \text{bird} \quad \text{white}$$

$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1)$$

USER

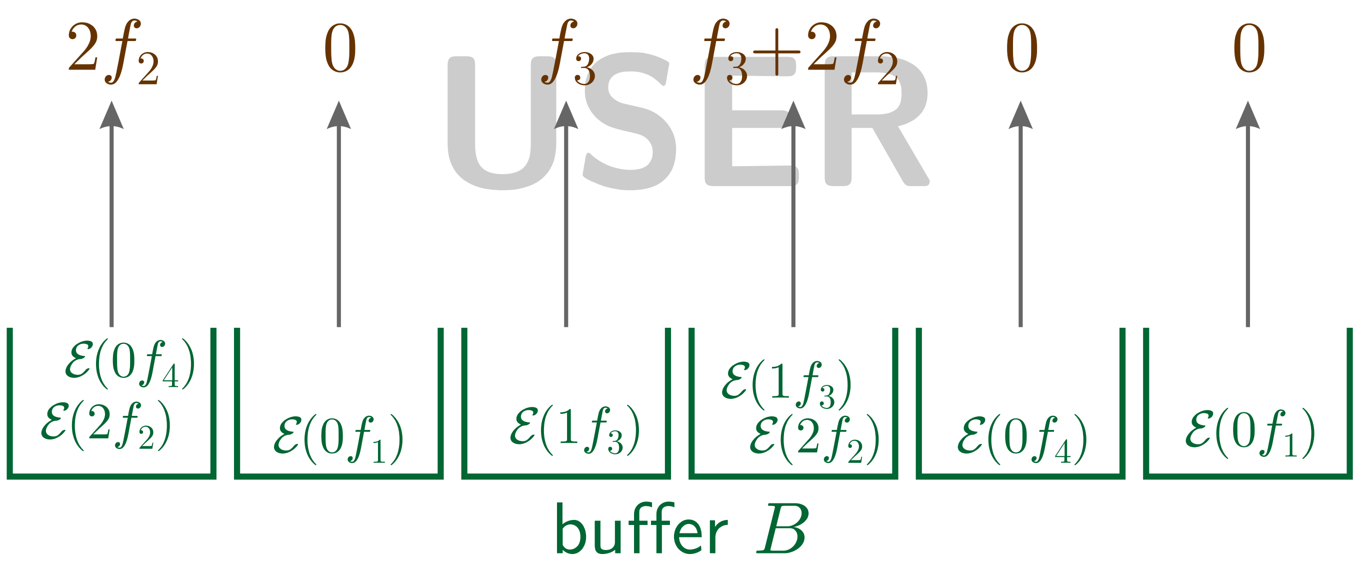
$\mathcal{E}(0f_4)$ $\mathcal{E}(2f_2)$	$\mathcal{E}(0f_1)$	$\mathcal{E}(1f_3)$	$\mathcal{E}(1f_3)$ $\mathcal{E}(2f_2)$	$\mathcal{E}(0f_4)$	$\mathcal{E}(0f_1)$

buffer B

- The user receives the encrypted buffer B.

$$\Omega = \text{dog} \quad \text{brown} \quad \text{cat} \quad \text{black} \quad \text{bird} \quad \text{white}$$

$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1)$$



- The user receives the encrypted buffer B,
  - he decrypts it.

$$\Omega = \text{dog} \quad \text{brown} \quad \text{cat} \quad \text{black} \quad \text{bird} \quad \text{white}$$

$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1)$$

"the cat is white" "the bird is white"

$$2f_2 \quad \cancel{0} \quad f_3 \quad \cancel{f_3 + 2f_2} \quad \cancel{0} \quad \cancel{0}$$

USER

$\mathcal{E}(0f_4)$ $\mathcal{E}(2f_2)$	$\mathcal{E}(0f_1)$	$\mathcal{E}(1f_3)$	$\mathcal{E}(1f_3)$ $\mathcal{E}(2f_2)$	$\mathcal{E}(0f_4)$	$\mathcal{E}(0f_1)$

buffer B

- The user receives the encrypted buffer B,
  - he gets one document for each singleton.

What can be improved in this scheme?

- Computations:
  - PSS requires one operation for each message,
  - difficult to improve,
    - → requires more efficient homomorphic encryption.
- Communications:
  - the query is linear in the dictionary size,
    - → fully homomorphic encryption could help,
  - the reply is linear in the buffer size,
    - → the buffer size should be the number of matches.
- In the Ostrovsky-Skeith scheme, the size is $O(m \log m)$,
  - Bethencourt *et al.* and Danezis-Díaz improve this.

Our Contribution

- Take an information theory look at the problem:
  - the server computes $\mathcal{E}(c_i f_i)$ an encrypted sparse vector
    - ⟶ the problem is to compress it,
  - possible to compute it's syndrome for any linear code
    - ⟶ compatible with homomorphic encryption.
- We propose two different approaches:
  - Using Reed-Solomon codes,
    - ⟶ allows a "zero-error" guarantee (if m is known),
    - ⟶ computationally heavy at the server,
  - Using irregular LDPC codes,
    - ⟶ gives optimal asymptotic performances.

Using LDPC codes

- To obtain an optimal reply size:
  - the user should only get the documents,
  - changing their order should not change the syndrome.

- Each document defines its own parity check column:
  - use the document as a seed to a PRNG,
  - use the PRNG to generate a “random” LDPC column.

⚠ This can’t be done in a standard communication,
→ we define the code from the values of the error.

PSS with LDPC codes

SERVER

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
1	0	0	0	0	1	1	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	0	0	1	1	0
0	0	1	0	0	1	0	1
1	0	0	0	1	0	1	1
0	0	0	1	1	0	0	1
0	1	0	1	0	0	0	0
0	0	1	0	1	0	0	0

- Use a PRNG to generate LDPC columns.

PSS with LDPC codes

$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1)$$

SERVER

$\mathcal{E}(0f_1)$	$\mathcal{E}(1f_2)$	$\mathcal{E}(1f_3)$	$\mathcal{E}(0f_4)$	$\mathcal{E}(1f_5)$	$\mathcal{E}(0f_6)$	$\mathcal{E}(0f_7)$	$\mathcal{E}(1f_8)$
1	0	0	0	0	1	1	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	0	0	1	1	0
0	0	1	0	0	1	0	1
1	0	0	0	1	0	1	1
0	0	0	1	1	0	0	1
0	1	0	1	0	0	0	0
0	0	1	0	1	0	0	0

- Compute the encrypted sparse vector $\mathcal{E}(c_i f_i)$ as before.

$$Q = \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1) \quad \mathcal{E}(0) \quad \mathcal{E}(0) \quad \mathcal{E}(1)$$

SERVER

$$\begin{array}{cccccccc} \mathcal{E}(0f_1) & \mathcal{E}(1f_2) & \mathcal{E}(1f_3) & \mathcal{E}(0f_4) & \mathcal{E}(1f_5) & \mathcal{E}(0f_6) & \mathcal{E}(0f_7) & \mathcal{E}(1f_8) \end{array}$$

$$\times$$

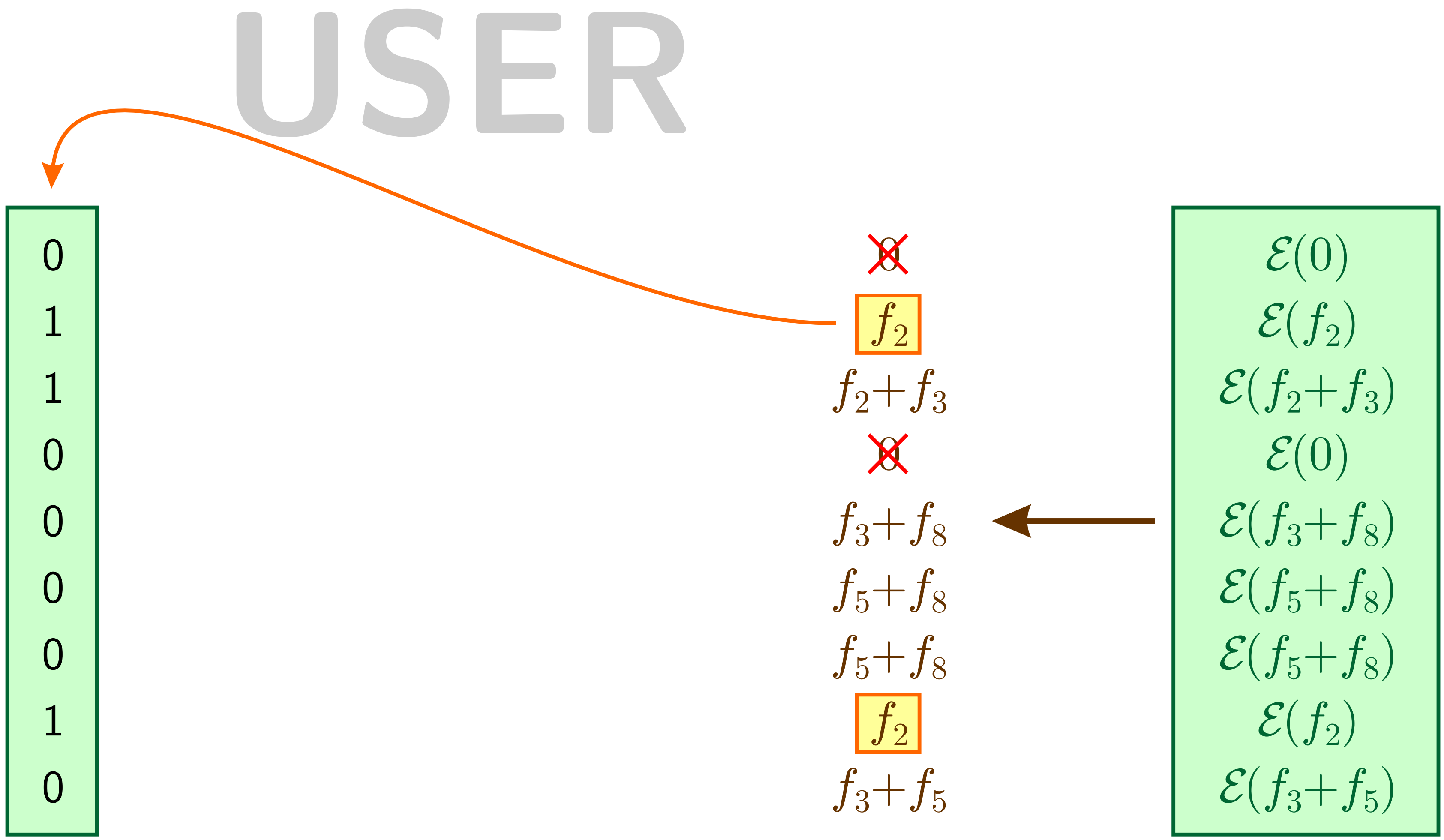
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \mathcal{E}(0) \\ \mathcal{E}(f_2) \\ \mathcal{E}(f_2+f_3) \\ \mathcal{E}(0) \\ \mathcal{E}(f_3+f_8) \\ \mathcal{E}(f_5+f_8) \\ \mathcal{E}(f_5+f_8) \\ \mathcal{E}(f_2) \\ \mathcal{E}(f_3+f_5) \end{bmatrix}$$

- Compute it's syndrome and send it to the user.

USER

$$
\begin{array}{ccc}
0 & & \mathcal{E}(0) \\
f_2 & & \mathcal{E}(f_2) \\
f_2+f_3 & & \mathcal{E}(f_2+f_3) \\
0 & & \mathcal{E}(0) \\
f_3+f_8 & \longleftarrow & \mathcal{E}(f_3+f_8) \\
f_5+f_8 & & \mathcal{E}(f_5+f_8) \\
f_5+f_8 & & \mathcal{E}(f_5+f_8) \\
f_2 & & \mathcal{E}(f_2) \\
f_3+f_5 & & \mathcal{E}(f_3+f_5)
\end{array}
$$

- The user first decrypts the buffer.



- For each singleton, he can generate its column.

USER

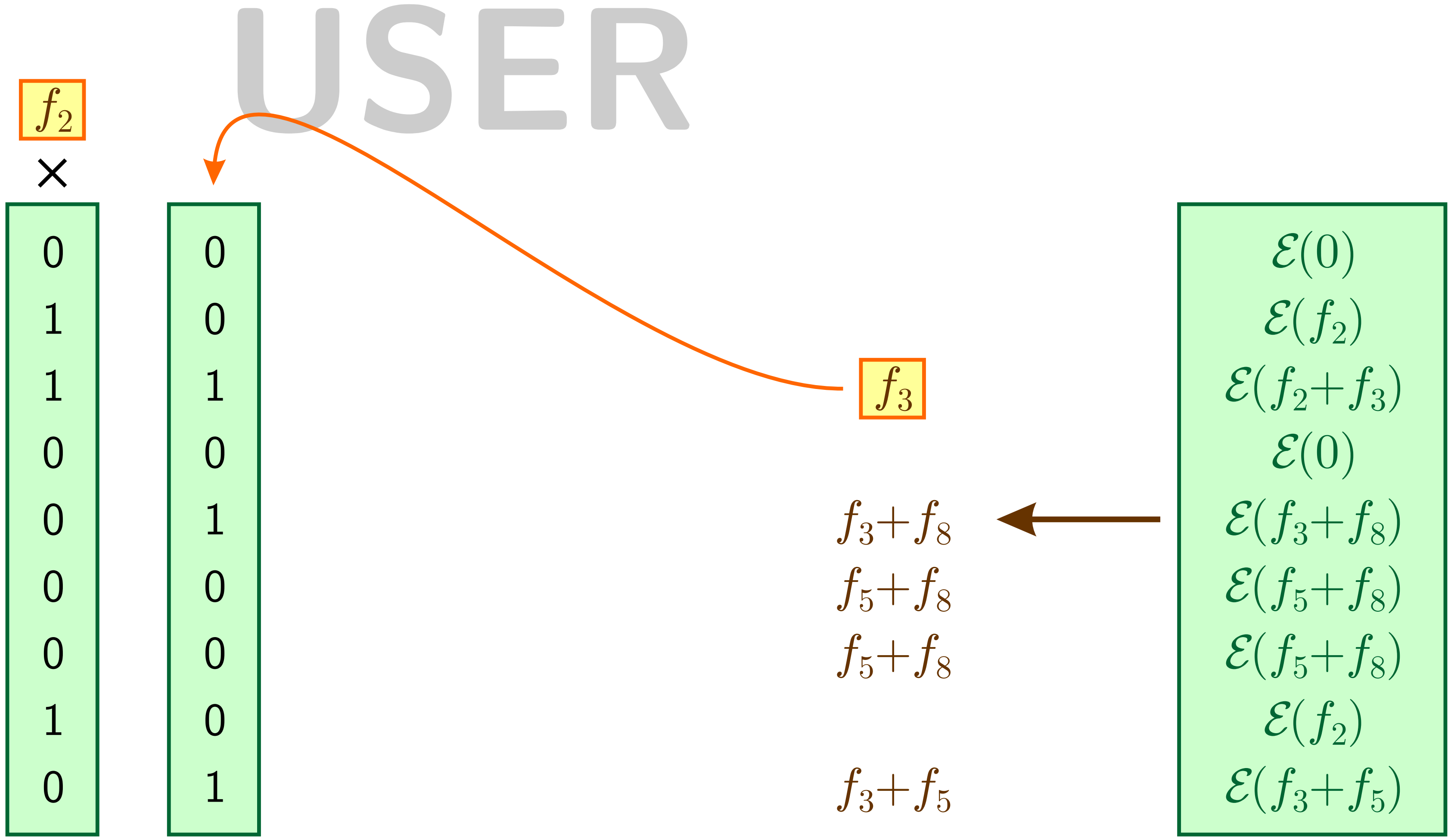
$\boxed{f_2}$

$\times$

0		$\mathcal{E}(0)$
1	$\cancel{f_2}$	$\mathcal{E}(f_2)$
1	$\cancel{f_2}+f_3$	$\mathcal{E}(f_2+f_3)$
0		$\mathcal{E}(0)$
0	f_3+f_8	$\mathcal{E}(f_3+f_8)$
0	f_5+f_8	$\mathcal{E}(f_5+f_8)$
0	f_5+f_8	$\mathcal{E}(f_5+f_8)$
1	$\cancel{f_2}$	$\mathcal{E}(f_2)$
0	f_3+f_5	$\mathcal{E}(f_3+f_5)$

- He can remove it completely from the buffer.

PSS with LDPC codes



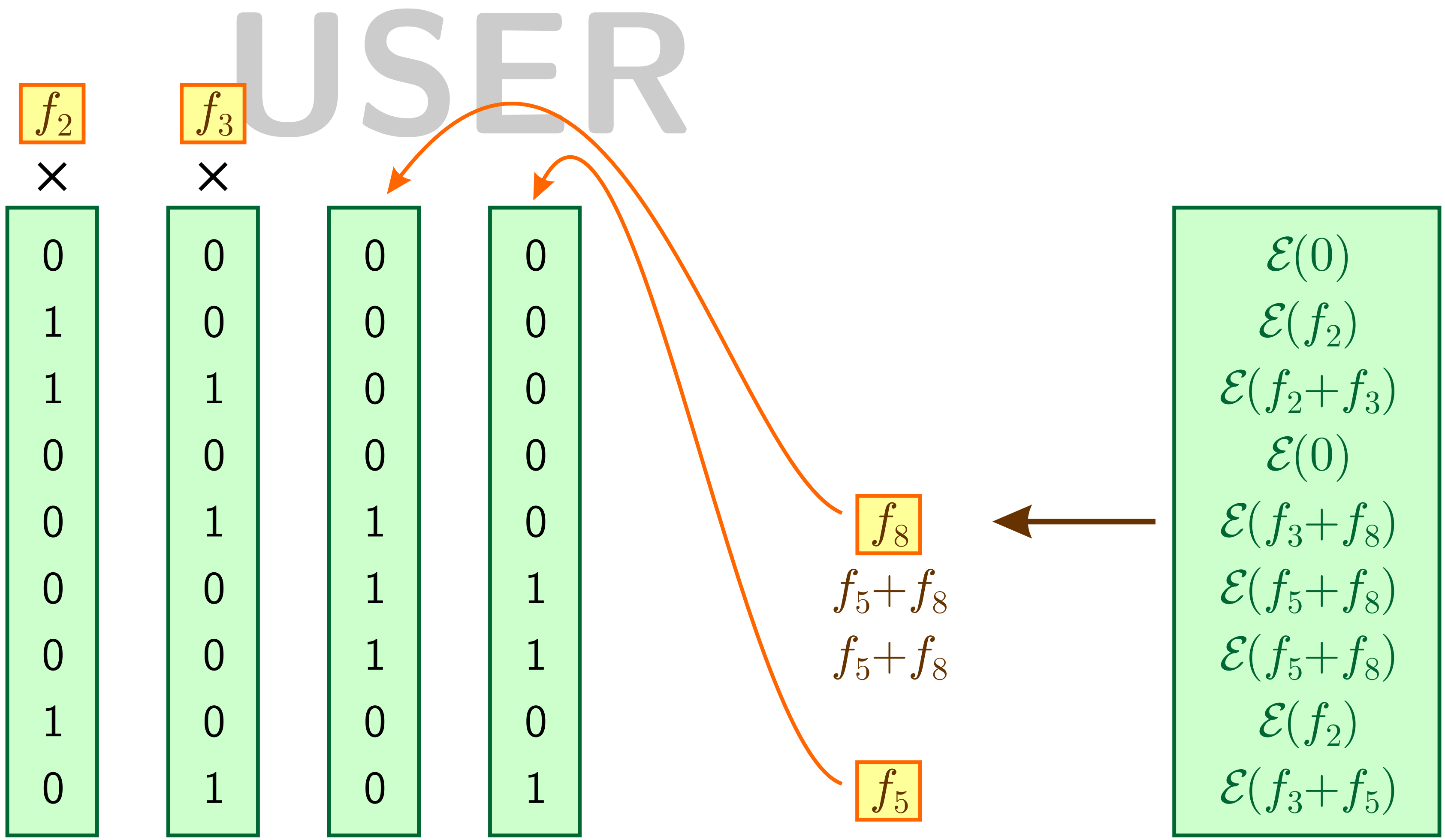
- This uncovers new singletons.

USER

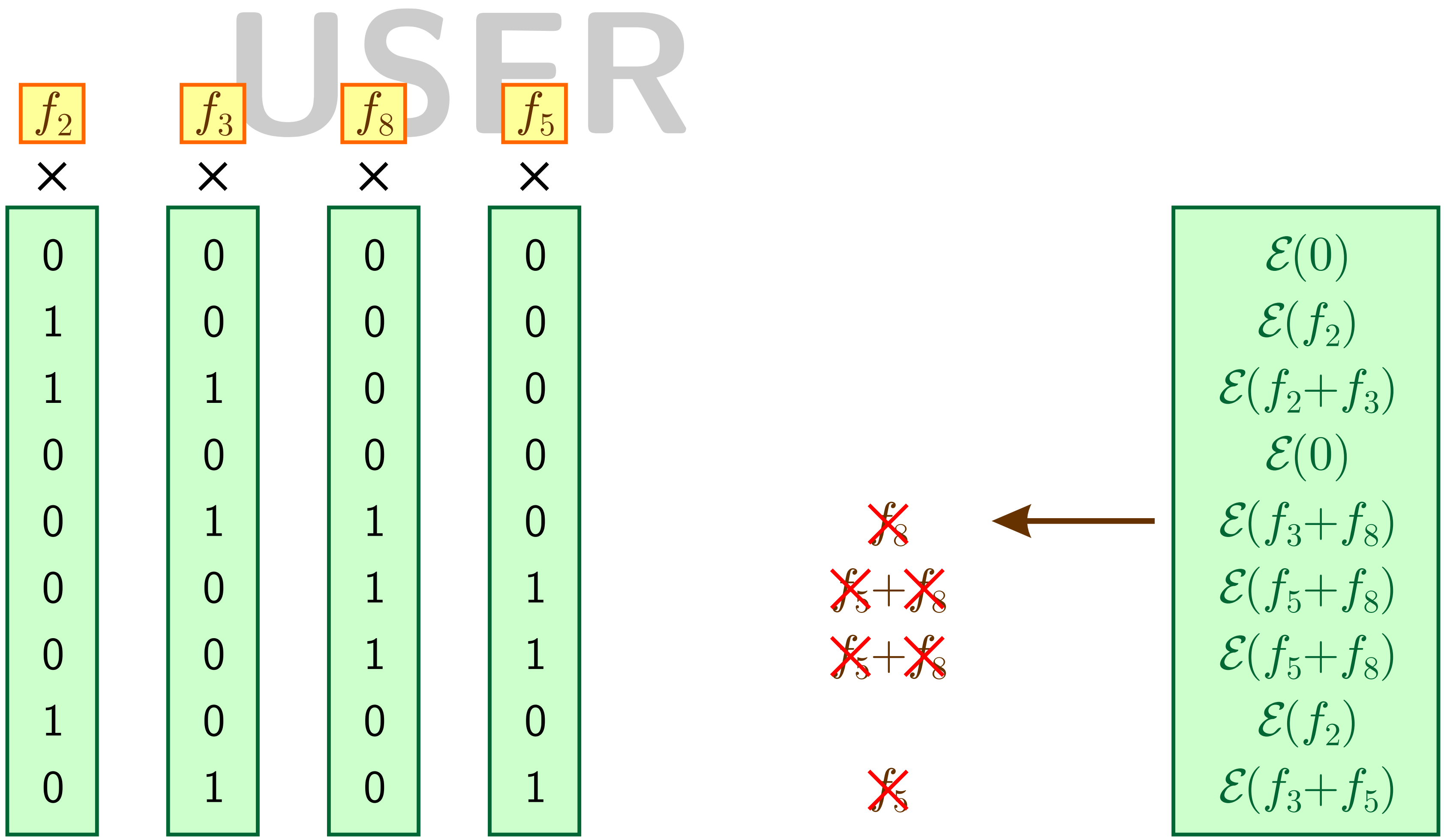
$\boxed{f_2}\times$	$\boxed{f_3}\times$		
0	0		$\mathcal{E}(0)$
1	0		$\mathcal{E}(f_2)$
1	1	$\cancel{f_3}$	$\mathcal{E}(f_2+f_3)$
0	0		$\mathcal{E}(0)$
0	1	$\cancel{f_3}+f_8$ $\longleftarrow$	$\mathcal{E}(f_3+f_8)$
0	0	f_5+f_8	$\mathcal{E}(f_5+f_8)$
0	0	f_5+f_8	$\mathcal{E}(f_5+f_8)$
1	0		$\mathcal{E}(f_2)$
0	1	$\cancel{f_3}+f_5$	$\mathcal{E}(f_3+f_5)$

- They can again be stripped from the buffer.

PSS with LDPC codes



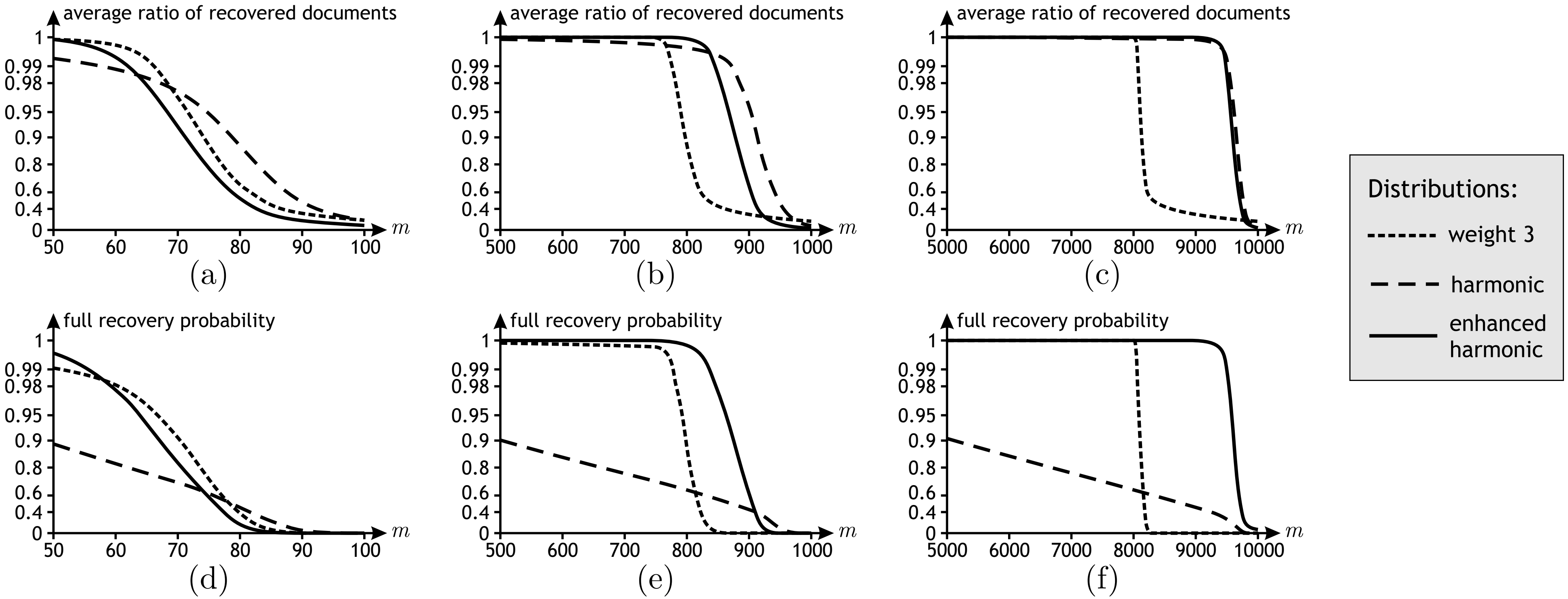
PSS with LDPC codes



- All documents were recovered when the buffer is 0.

- The whole algorithm is independent of the stream size,
  - the buffer size depends only on the number of matches.
- Computationally very efficient:
  - for the server, one “encryption” per document,
  - for the user, one decryption per buffer position
    → the rest of the decoding is also linear.
- We have full control on the column distribution,
  - possible to use constant weight,
    → not optimal asymptotically,
  - possible to use irregular LDPC codes,
    → use work of Luby, Mitzenmacher, Shokrollahi for asymptotic analysis.

PSS with LDPC codes
Simulation results



- Simulations for buffers of sizes 100, 1 000 and 10 000:
  - for 100, constant weight is as good as irregular,
  - we see the asymptotic limitation of constant weight
    $\longrightarrow$ at least a ratio 1.22 between buffer/matches.

Conclusion

- Our new Private Stream Search scheme:
  - compared to the Ostrovsky-Skeith scheme
    $\longrightarrow$ same computational cost, better communication,
  - compared to a non-private search
    $\longrightarrow$ same asymptotic communication cost, additional computations (especially for the server)
- Is it practical?
  - probably too expensive using Paillier's encryption
    $\longrightarrow$ lighter homomorphic encryption (lattice based?).
  - practical from a communication point of view.
- Would any search engine want to use it?