CCS' 18

Secure Search
on Encrypted Data via Multi-Ring Sketch

Adi Akavia, Dan Feldman, and Hayim Shaul

Definition 1 (Secure Search). *The server holds an unsorted array of encrypted values (previously uploaded to the server, and where the server has no access to the secret decryption key):* Searched data √

$$[\![array]\!] = ([\![x_1]\!], \ldots, [\![x_m]\!])$$

(here and throughout this work, $[\![msg]\!]$ denotes the ciphertext encrypting message msg; the encryption can be any fully, or leveled, homomorphic encryption (FHE) scheme, e.g. [7]). The client sends to the server an encrypted lookup value $[\![\ell]\!]$. The server returns to the client an encrypted index and value

Search query √ Data access pattern √

$$[\![y]\!] = ([\![i]\!], [\![x_i]\!])$$

satisfying the condition:

$$is\textsc{Match}(x_i, \ell) = 1$$

for $is\textsc{Match}()$ a predicate specifying the search condition (see discussion below on using generic predicates). More generally, y may be a value from which the client can compute (i, x_i) (decode).

We call the client efficient *if its running time is polynomial in the output length $|i| = O(\log m)$ and $|x_i|$ and in the time to encrypt/decrypt a single ciphertext. The* server is efficient *if the polynomial $f([\![array]\!], [\![\ell]\!])$ the server evaluates to obtain $[\![y]\!]$ is of degree polynomial in* $\log m$ *and the degree of $is\textsc{Match}()$, and of size (i.e., the overall number of addition and multiplication operations for computing f) polynomial in m and the size of $is\textsc{Match}$. The* protocol is efficient *if both client and server are efficient. (We call the client/server/protocol* inefficient *if the running time/degree/either is at least $\Omega(m)$.)*

Prior Works

Protocols and Papers	Efficient Client	Efficient Server	Supports unrestricted search functionality	Retrieves record	Full security	Records per hour per machine
Searchable Encryption [6,42]	✓	N/A	✓	✓	× Leak information	Gb
PIR [16,8,14,9,40]	✓	✓	× UNIQUE	✓	✓	Mb
PSI [10]	✓	✓	× UNIQUE	× Decision problem	✓	Mb
Secure Pattern Matching [47,13,32,11,21,27]	× $\Omega(m)$	✓	✓	∼ ✓	✓	Kb
Folklore Secure Search	✓	× $\Omega(m)$	✓	✓	✓	Kb
This work: Secure Search	✓	✓	✓	✓	✓	Mb

Table 2. *Comparison to single-round secure search protocols.* 1st column lists the compared works, followed by indications to whether: client and server are efficient (✓) or inefficient (×) in columns 2-3; the scheme supports unrestricted search functionality (✓) or requires a unique identifier (×) in column 4; the client's output is both index and record (✓), only an index i (∼ ✓), or only a YES/NO answer to whether the record exists (×), in column 5; the scheme is fully secure (✓) in the sense of attaining semantic security for the data and lookup value both at rest and during search, as well as hiding the access pattern to the database, in column 6. Last column specifies number of processed records per hours per machine in reported experiments: thousands (Kb), millions (Mb), billions (Gb).

Multi ring FHE

Requirement: plaintext modulus to be a prime number p of our choice

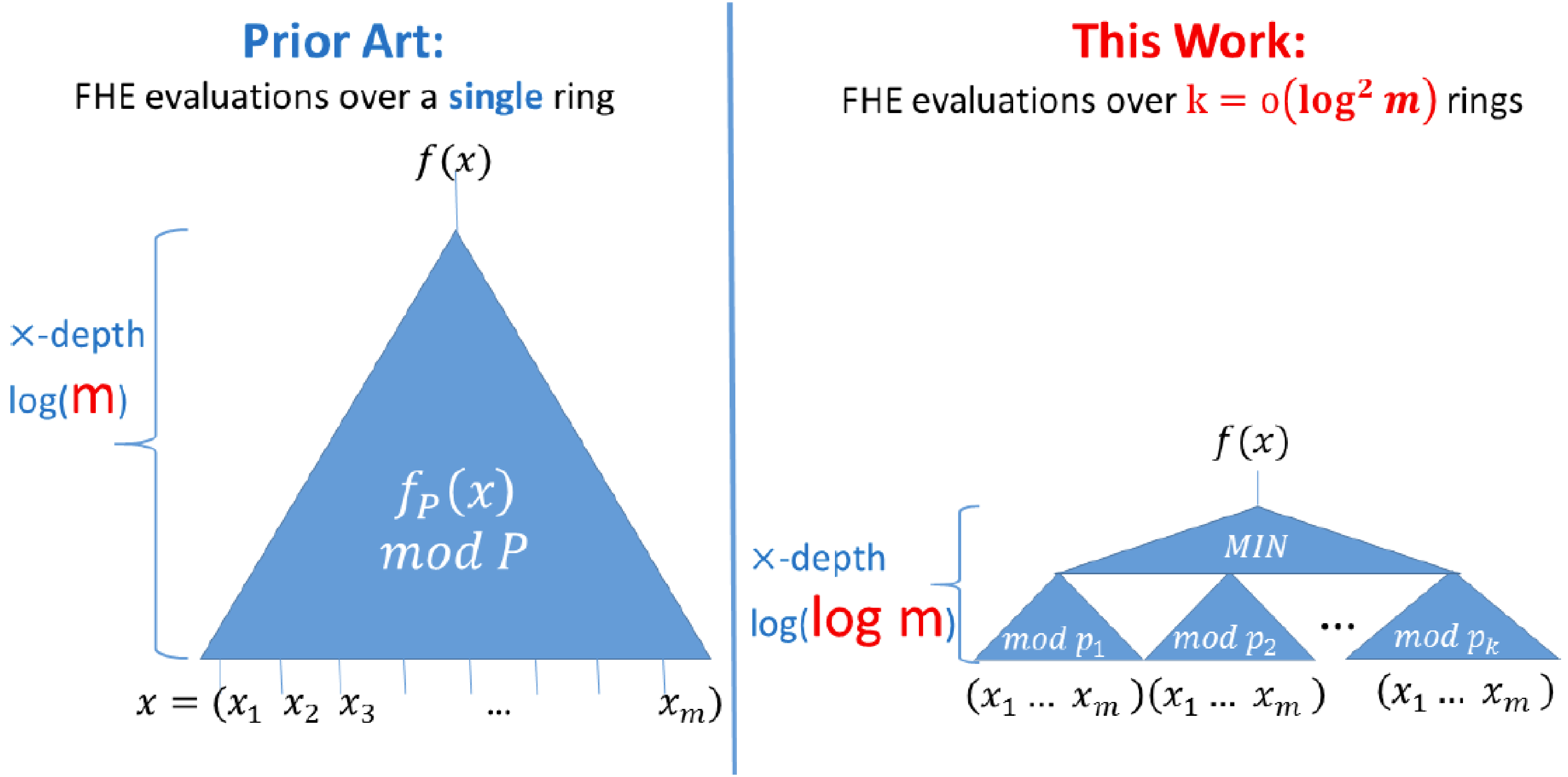

Fig. 3. Single/Multi ring arithmetic circuit for secure search: the multiplicative depth of known single ring circuits is exponentially higher than in our proposed multi ring circuit.

Multi ring FHE

- FHE scheme $E = (Gen, Enc, Dec, Eval)$:
  - Gen: randomized algorithm
    - Input: security parameter λ, prime p
    - Output: secret key $sk_p = (p, sk)$, evaluation key $ek_p = (p, ek)$ for plaintext modulus p
    - $\left(sk_p = (p, sk), ek_p = (p, ek)\right) \leftarrow Gen(1^\lambda; p)$

Multi ring FHE

- FHE scheme $E = (Gen, Enc, Dec, Eval)$:
  - Enc: randomized algorithm
    - Input: secret key sk_p, plaintext message msg
    - Output: ciphertext $[\![msg]\!]_p$ for plaintext modulus p
    - $[\![msg]\!]_p \leftarrow Enc_{sk_p}(msg)$

Multi ring FHE

- FHE scheme $E = (Gen, Enc, Dec, Eval)$:
  - Dec
    - Input: secret key sk_p, ciphertext $[\![msg]\!]_p$
    - Output: plaintext msg'
    - $msg' \leftarrow Dec_{sk_p}\left([\![msg]\!]_p\right)$
    - Correctness: $msg' = msg$

Multi ring FHE

- FHE scheme $E = (Gen, Enc, Dec, Eval)$:
  - $Eval$: possibly randomized algorithm
    - Input: evaluation key $ek_p = (p, ek)$ for plaintext modulus p, polynomial $f(x_1, \dots, x_t)$, tuple of ciphertexts $\left(⟦m_1⟧_p, \dots, ⟦m_t⟧_p\right)$
    - Output: ciphertext c
    - $c \leftarrow Eval_{ek_p}\left(f, ⟦m_1⟧_p, \dots, ⟦m_t⟧_p\right)$
    - Correctness: $Dec_{sk_p}\left(Eval_{ek_p}\left(f, ⟦m_1⟧_p, \dots, ⟦m_t⟧_p\right)\right) = f(m_1, \dots, m_t) \bmod p$
    - Semantic security: ciphertext c is computationally indistinguishable from a fresh ciphertext $⟦f(m_1, \dots, m_t)⟧_p$

Uploading Encrypted Data

Secure outsourcing of computation

Algorithm 1: Data Upload Protocol

Shared Input: An FHE scheme $E = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Eval})$,
A number m of data records in *array*, where w.l.o.g. we assume m is a power of two,
A set $\mathcal{P} = \{p_1, \ldots, p_k\}$ of the smallest $k = 1 + \log^2 m$ primes that are larger than $\log m$.

Inputs: The client's input is a security parameter λ and $array = (array(1), \ldots, array(m))$
The server has no input.

Outputs: The client's output is a secret key $sk = (sk_{p_1}, \ldots, sk_{p_k})$ (for the FHE and security λ).
The server's output is the corresponding evaluation key $ek = (ek_{p_1}, \ldots, ek_{p_k})$, and the encrypted data $[\![array]\!] = ([\![array]\!]_{p_1}, \ldots, [\![array]\!]_{p_k})$
(where *array* is encrypted entry-by-entry: $[\![array]\!]_p = ([\![array(1)]\!]_p, \ldots, [\![array(m)]\!]_p)$).

The client encrypt the data and sends to the server

1. The client does the following:
   - Generate keys $(sk_{p_1}, ek_{p_1}) \leftarrow \mathrm{Gen}(1^\lambda; p_1), \ldots, (sk_{p_k}, ek_{p_k}) \leftarrow \mathrm{Gen}(1^\lambda; p_k)$. Denote

$$ek = (ek_{p_1}, \ldots, ek_{p_k}).$$

   - Compute for all $i \in [m]$ and $j \in [k]$:

$$[\![array(i)]\!]_{p_j} \leftarrow \mathrm{Enc}_{sk_{p_j}}(array(i)).$$

   - Send to server

$$ek \text{ and } [\![array]\!] = ([\![array]\!]_{p_1}, \ldots, [\![array]\!]_{p_k}),$$

   where *array* is encrypted entry by entry: $[\![array]\!]_p = ([\![array(1)]\!]_p, \ldots, [\![array(m)]\!]_p)$.

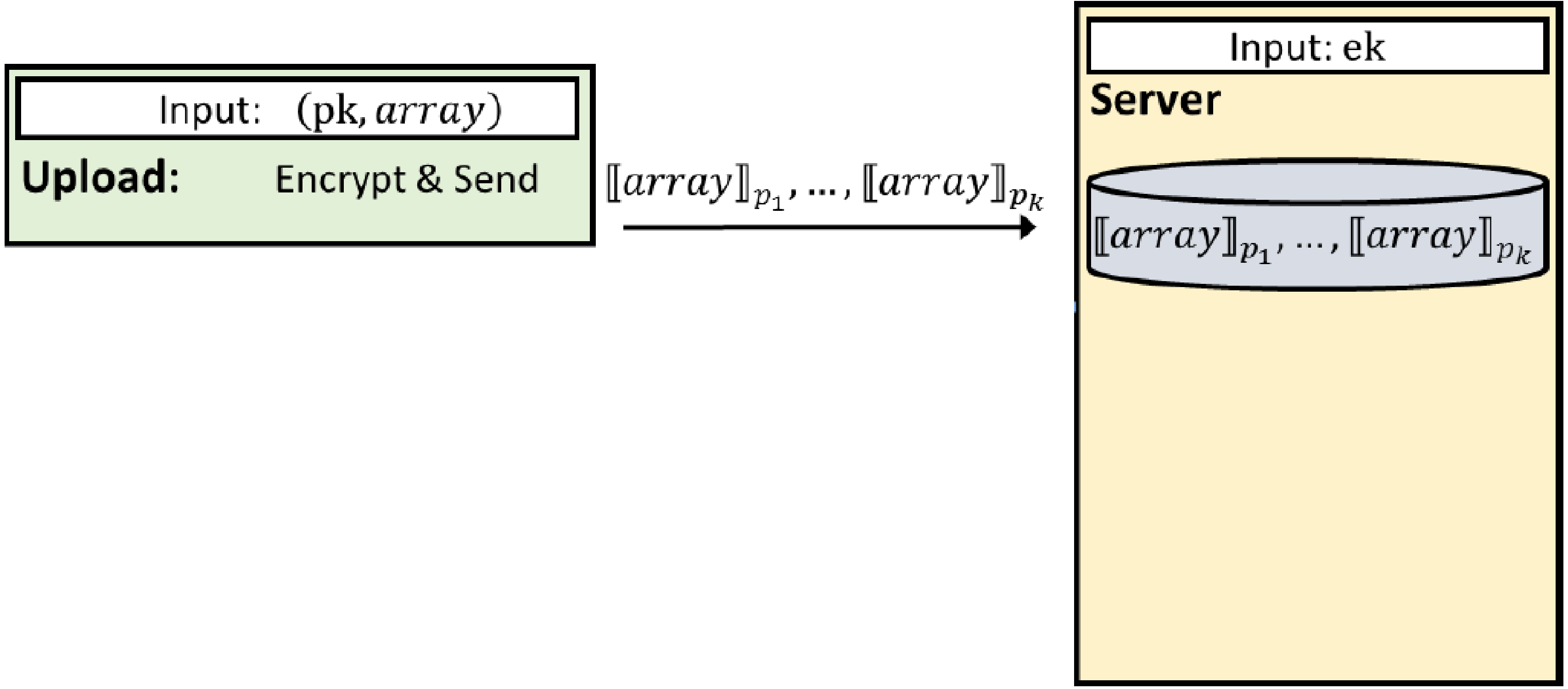

Fig. 2. Depicting our Secure Search Protocol 2 and the Data Upload Protocol 1. In the data upload protocol the client, whose input is the public key and the data *array*, encrypts the data and sends to the server. In the secure search protocol, the server's input is the evaluation key and the encrypted data that was previously uploaded; The client's input is the secret key and a lookup value; A common input (in both Protocols 1-2) is the set of prime numbers $p_1, \ldots, p_k$. The client sends to the server the lookup value ℓ encrypted in k ciphertexts with plaintext moduli $p_1, \ldots, p_k$; where we use the notation $[\![x]\!]_p$ to denote a ciphertext encrypting message x to enable homomorphic addition and multiplication modulo p. The server evaluates, for each modulus p_j, the pattern matching polynomial $is\textsc{Match}$ on the encrypted lookup value $[\![\ell]\!]_{p_j}$ and data $[\![array]\!]_{p_j}$ to obtain an encrypted indicator vector $[\![ind]\!]_{p_j}$. The server then evaluates on this encrypted indicator vector our SPiRiT_{m,p_j} sketch for first positive to obtain a candidate $[\![b_{p_j}]\!]_{p_j}$ for its first positive index. The server sends to the client these k candidates $[\![b_j]\!]_{p_j}$ together with the corresponding k entries in ind. The client decrypts and outputs the smallest candidate b_j s.t. $ind(b_j) = 1$.

The Secure Search Protocol

Algorithm 2: Secure Search Protocol

Shared Input: An FHE scheme $E = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Eval})$,
A power of two m denoting the number of data records in *array*,
A set $\mathcal{P} = \{p_1, \dots, p_k\}$ of the smallest $k = 1 + \log^2 m$ primes that are larger than $\log m$.
A pattern matching polynomial $is\textsc{Match}(\cdot,\cdot)$.

Inputs: Client's input is the secret key $sk = (sk_{p_1}, \dots, sk_{p_k})$ and a lookup value ℓ.
The server's input is the corresponding evaluation key $ek = (ek_{p_1}, \dots, ek_{p_k})$, and the encrypted data $[\![array]\!] = ([\![array]\!]_{p_1}, \dots, [\![array]\!]_{p_k})$.

Outputs: The client's output is (the binary representation $b^* \in \{0,1\}^{1+\log m}$ of) the index $i^* = \min\{ i \in [m] \mid is\textsc{Match}(array(i), \ell)\}$.
The server has no output.

1. The client compute for all $j \in [k]$:
$$[\![\ell]\!]_{p_j} \leftarrow \mathrm{Enc}_{sk_{p_j}}(\ell).$$
and sends to the server
$$([\![\ell]\!]_{p_1}, \dots, [\![\ell]\!]_{p_k}),$$
2. The server does the following for each $j \in [k]$:
   (a) Compute
$$[\![indicator]\!]_{p_j} \leftarrow (is\textsc{Match}([\![array(1)]\!]_{p_j}, [\![\ell]\!]_{p_j}), \dots, is\textsc{Match}([\![array(m)]\!]_{p_j}, [\![\ell]\!]_{p_j})).$$
   The (encrypted) binary vector *indicator* with 1 in all entries of *array* that match *l* using a generic *isMatch*() pattern matching protocol
   (b) Compute
$$[\![b_{p_j}]\!] \leftarrow \mathrm{SPiRiT}_{m,p_j}([\![indicator]\!]_{p_j})$$
   where $\mathrm{SPiRiT}_{m,p_j} = S \circ P \circ i \circ R \circ i \circ T$ for S, P, R, T and i the matrices and operator specified in Section 3.4 below. A short list of candidates for the index i^* of the first positive entry in *indicator* using our *SPiRiT* sketch for first positive
   (c) Send to the client
$$([\![b_{p_1}]\!], \dots, [\![b_{p_k}]\!]) \text{ and } ([\![indicator(b_{p_1})]\!]_{p_1}, \dots, [\![indicator(b_{p_k})]\!]_{p_k})$$
   (here we slightly abuse notation by addressing entries of *indicator* by the binary representation of the indices). To compute $[\![indicator(b_{p_j})]\!]_{p_j}$ the server applies standard PIR techniques, namely, evaluating *indicator* and b_j the polynomial $\sum_{i=1}^{m} indicator(i) \cdot is\textsc{Equal}(i, b_{p_j})$; see Section 3.5.
3. The client decrypts and outputs the minimum The client decodes and chooses the smallest of the verified candidates as the output
$$b^* \leftarrow \min_{j \in [k]} \{ b_{p_j} \text{ s.t. } indicator(b_j) = 1\}.$$

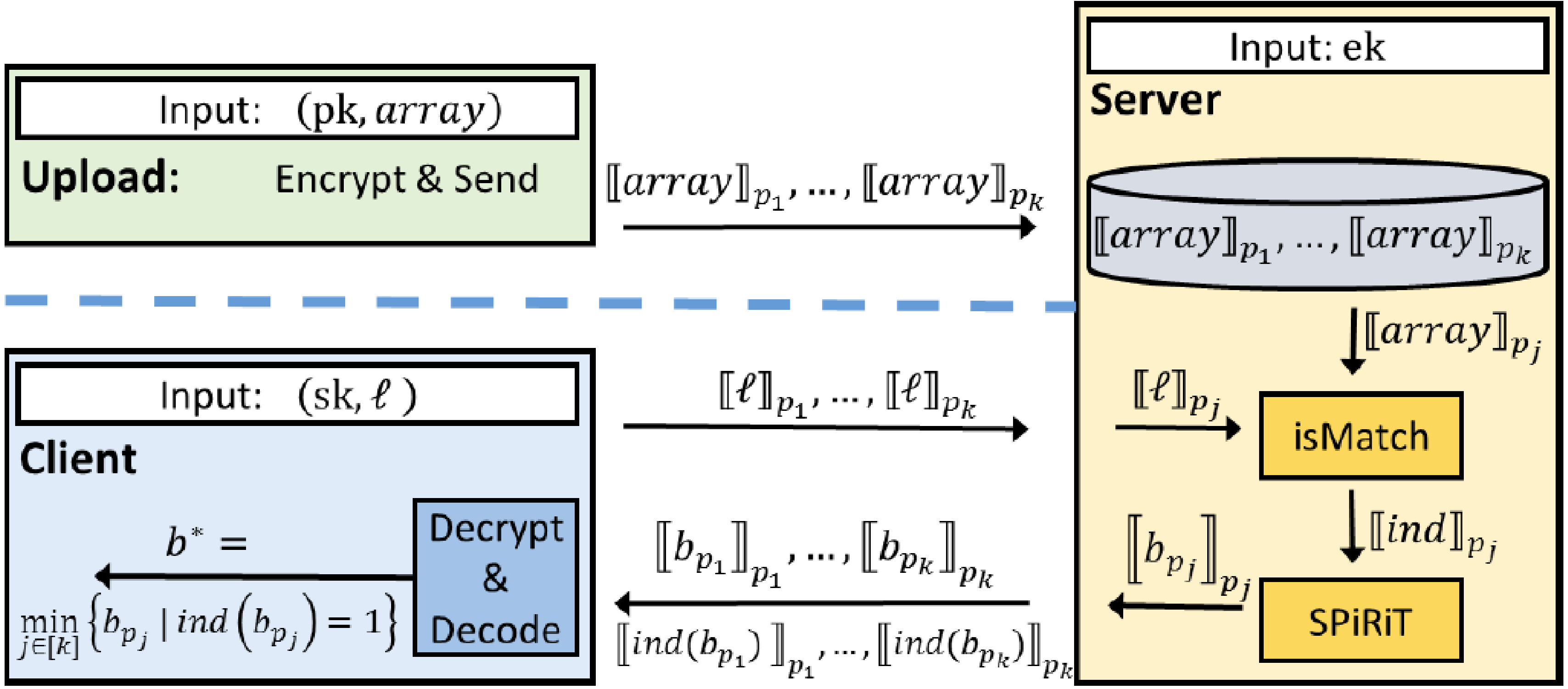

Fig. 2. Depicting our Secure Search Protocol 2 and the Data Upload Protocol 1. In the data upload protocol the client, whose input is the public key and the data *array*, encrypts the data and sends to the server. In the secure search protocol, the server's input is the evaluation key and the encrypted data that was previously uploaded; The client's input is the secret key and a lookup value; A common input (in both Protocols 1-2) is the set of prime numbers $p_1, \ldots, p_k$. The client sends to the server the lookup value ℓ encrypted in k ciphertexts with plaintext moduli $p_1, \ldots, p_k$; where we use the notation $[\![x]\!]_p$ to denote a ciphertext encrypting message x to enable homomorphic addition and multiplication modulo p. The server evaluates, for each modulus p_j, the pattern matching polynomial $is\textsc{Match}$ on the encrypted lookup value $[\![\ell]\!]_{p_j}$ and data $[\![array]\!]_{p_j}$ to obtain an encrypted indicator vector $[\![ind]\!]_{p_j}$. The server then evaluates on this encrypted indicator vector our SPiRiT_{m,p_j} sketch for first positive to obtain a candidate $[\![b_{p_j}]\!]_{p_j}$ for its first positive index. The server sends to the client these k candidates $[\![b_j]\!]_{p_j}$ together with the corresponding k entries in ind. The client decrypts and outputs the smallest candidate b_j s.t. $ind(b_j) = 1$.

Sketch for First Positive

Sketch for First Positive

- **Hard problem:** single output from a large-ring polynomial
  - The evaluation of $SPiRiT_{m,p}$ on a single large ring $p = \Omega(m)$
- **Easier problem:** multiple outputs from few small rings polynomial
  - Few $k = o(\log^2 m)$ evaluations of polynomials but on small rings moduli $p_1, \dots, p_k = O(\log^2 m)$
  - Evaluate in parallel k polynomials each of low degree $O(\log^4 m)$
- **Cost:** additional but minor amount of computation on the client side

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- (indicator) Return the step function $u \in \{0,1\}^m$ accepting value 0 on entries $1, \dots, i^* - 1$ and value 1 on entries $i^*, \dots, m$
  - The **T**ree matrix T: the labels of a binary tree
    - m leaves labeled by the entries of indicator
    - Each node: labeled by the sum of the labels of its children
  - $\boldsymbol{isPositive_p}$ operator: reduce to binary values
  - The **R**oot matrix R:
    - Partition the tree leaves $1, \dots, i$ according to their deepest common ancestor whose leaves are contained in the leaves $1, \dots, i$
    - Sum up the labels of these ancestors
  - $\boldsymbol{isPositive_p}$ operator: reduce to binary values

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- The step function u: binary vector with a single non-zero entry at index i^*
  - The **P**airwise difference matrix P: compute the derivative of u
  - The **S**ketch matrix $S \in \{0,1\}^{(1+\lceil \log m \rceil)\times m}$:
    - Standard sketch matrix for 1-sparse vectors
      - A matrix that given a binary vector with at most a single non-zero entry returns the binary representation of the index of this entry (or zero if none exists)
  - $\boldsymbol{isPositive_p}$ operator: [Fermat's Little Theorem]
    - $\boldsymbol{isPositive_p}(x_1, \dots, x_{m'}) = \left(x_1^{p-1} \bmod p, \dots, x_{m'}^{p-1} \bmod p\right)$

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- The **T**ree matrix $T \in \{0,1\}^{(2m-1)\times m}$
  - Right multiplication by a length m vector $x = (x_1, \dots, x_m)$
  - Return the length $2m-1$ array data structure data structure representation $w = (w_1, \dots, w_{2m-1})$ for the tree representation $\mathcal{T}(x)$ of x

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- The Tree representation $\mathcal{T}(x)$ of x: the full binary tree of depth $\log_2 m$
  - The label of the ith leftmost leaf: $x(i)$, for $i \in [m]$
  - The label of each inner node: the sum of the labels of its two children
  - The array data structure: vector $w = \big(w(1), \dots, w(2m-1)\big)$
    - $w(1)$: the label of the root
    - $w(2j)$, $w(2j+1)$: the labels of the left and right children of the node whose label is $w(j)$, for every $j \in [m-1]$
    - $w(i)$: the sum of the labels of the leaves of the subtree rooted in the tree node corresponding to array entry $w(i)$

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- The Tree matrix T: $w = Tx$
  - Each row k of T corresponds to the node u in $\mathcal{T}(x)$ represented by $w(k)$
  - This row has 1 in every column j so that u is an ancestor of the jth leaf (0 otherwise)
- The Tree matrix T
  - Independent of x
  - The last m entries of w are the entries of $x = \big(w(m), \dots, w(2m-1)\big)$

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- The Roots matrix $R \in \{0,1\}^{m \times (2m-1)}$
  - Each row has $O(\log m)$ non-zero entries
  - Right multiplications by the tree representation $w = \big(w(1), \dots, w(2m -$

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- Naïve implementation: $R' \in \{0,1\}^{m\times m}$ whose ith row $(1,\dots,1,0,\dots,0)$ consists of i ones followed by $m-1$ zeros for every $i \in [m]$
  - Do not satisfy requirement for $O(\log m)$ non-zero entries in each row
    - Even when applies on binary vectors the result may consist of values up to m
    - Ruin the success of $SPiRiT_{m,p}$ when using small primes $p \ll m$

- 存疑
  - In-place 遍历?

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- For each node $j \in [2m-1]$
  - $Ancestors(j) \subseteq [2m-1]$: the set of indices corresponding to the ancestors in the tree of node j (including j itself)
  - $Siblings(j) \subseteq [2m-1]$: the set of indices corresponding to the left-siblings of $Ancestors(j)$
- The Roots matrix R
  - Sum labels of only $O(\log m)$ internal nodes of the tree representation w of x
  - Each row i of the matrix R has values 1 in all entries $j \in Siblings(j+1)$ (0 otherwise)

$$R(i,j) = \begin{cases} 1, & if\ i \in [m]\ and\ j \in Siblings(i+1) \\ 0, & otherwise \end{cases}$$

$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$

Lemma 1 (Roots sketch). *Let* $T \in \{0,1\}^{2m-1 \times m}$ *and* $R \in \{0,1\}^{m \times (2m-1)}$ *be the matrices defined above. Then each row of* R *has at most* $\log m$ *non-zero entries, and for every* $x = (x(1), \ldots, x(m))$*, the vector* $v = RTx$ *is a length* m *vector so that for every* $j \in [m]$*,*

$$v(j) = \sum_{k=1}^{j} x(k).$$

- Property
  - Each row having $O(\log m)$ non-zero entries
  - $v = RTx$ being the vector of prefix sums of x

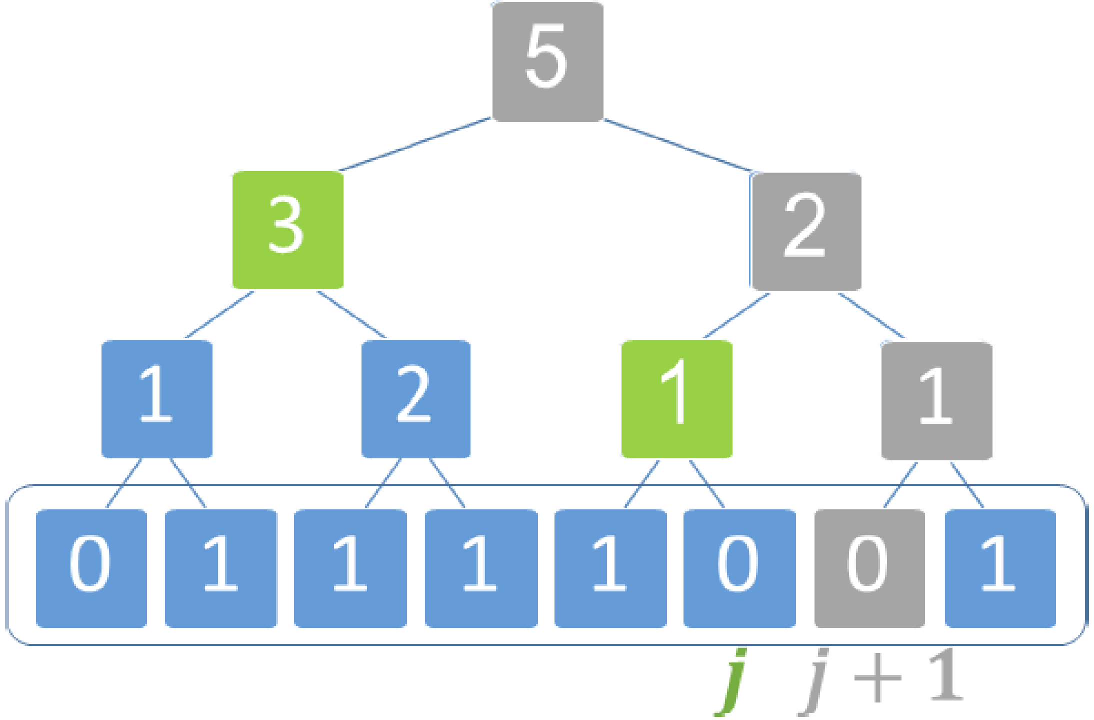

Fig. 4. The tree representation for a length $m = 8$ binary vector $indicator = (0, 1, 1, 1, 1, 0, 0, 1)$ is the full binary tree with m leaves labeled by entries of $indicator$, and with internal nodes labeled by the sums of their children's labels. The array data structure for this tree is the length $2m - 1$ vector $w = T \cdot indicator = (5, 3, 2, 1, 2, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1)$. The prefix sum of leaves' labels up to the $j = 6th$ leaf from the left is $v(6) = 0 + 1 + 1 + 1 + 1 + 0 = 4$. More generally, the vector of prefix sums of $indicator$ is $v = (0, 1, 2, 3, 4, 4, 4, 5)$. The root matrix R has the property that $RT \cdot indicator = Rw = v$. To construct sparse R we observe that every entry of v can be computed using only $O(\log m)$ labels; this is by summing the roots of subtrees forming a partition of the leaves in the considered prefix. For example, to compute the $j = 6th$ entry $v(6)$ we sum two labels as follows: First identify all the ancestors of the $j + 1 = 7th$ leaf: labeled by $0, 1, 2, 5$ in the figure (colored in grey). Among these ancestors, select those who are right children: labeled by 1 and 2 in the figure. Finally, sum the labels of the left siblings of these selected ancestors: labeled by 1 and 3 in the figure (colored in green) to get the desired sum. Indeed, $v(6) = 1 + 3 = 4$. .

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- The Pairwise matrix $P \in \{-1, 0, 1\}^{m \times m}$
  - Right multiplication by a given length m vector $u = \big(u(1), \dots, u(m)\big)$
  - Yield the vector $t = Pu$ of pairwise differences between consecutive entries in u
    - $t(j) = u(j) - u(j-1)$ for every $j \in \{2, \dots, m\}$ and $t(1) = u(1)$
  - Every row $i \in \{2, \dots, m\}$: the form $(0, \dots, -1, 1, \dots, 0)$ with 1 appearing at its i-th entry
    - The first row is $(1, 0, \dots, 0)$

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- The Sketch matrix $S \in \{0,1\}^{(1+\log m)\times m}$
  - Right multiplication by a binary vector $t = (0, \dots, 0, 1, 0, \dots, 0) \in \{0,1\}^m$ with a single non-zero entry in its jth coordinate
  - Yield the binary representation $y = St \in \{0,1\}^{1+\log m}$ of $j \in [m]$
    - $y = 0^{1+\log m}$ if t is the all zero vector
  - Each column $j = \{1, \dots, m\}$: the binary representation of j

$$SPiRiT_{m,p} = S \circ P \circ i \circ R \circ i \circ T$$

- The isPositive operator $i()$
  - Input an integer vector $v = (v_1, \dots, v_{m'})$
  - Return a binary vector $u \in \{0,1\}^{m'}$
    - For every $j \in [m']$, $u(j) = 0$ if and only if $v(j)$ is a multiple of p [Fermat's Little Theorem]
      $\boldsymbol{isPositive}\big(v(1), \dots, v(m')\big) = (v(1)^{p-1}\ mod\ p, \dots, v(m')^{p-1}\ mod\ p)$

Problem

Correctness

- Require a large modulus $p = \Omega(m)$
  - Wrong outputs due to overflow

Efficiency

- Require a small modulus $p = \log^{O(1)} m$
  - The degree of $SPiRiT_{m,p}$ is polynomial in p

Key Property: *If the labels of the ancestors of the first positive leaf i^* in the tree $T(indicator)$ are not multiples of p, then $\mathrm{SPiRiT}_{m,p}(indicator)$ returns the binary representation of i^**; see Lemma 3, Section 4.1.

- At most $\log^2 m$ primes p that are divisors of these labels
- Pigeonhole principle
  - For any $k = 1 + \log^2 m$ primes p, at least one of them would satisfy the above condition on the ancestors of i^*
- Compute $SPiRiT_{m,p}$(indicator) (in parallel) for k primes p
  - A list of candidates $b_1, \dots, b_k$ for the binary representation b^* of the first positive index i^*, with the guarantee that for $i_1, \dots, i_k \in [m] \cup \{0\}$ the corresponding indices

$$i^* = \min_{j \in [k]} \{ i_j \ \mathrm{s.t.}\ indicator(i_j) = 1 \}$$

The Secure Pattern Matching

The Secure Pattern Matching - Example

- The equality test: $isMatch(array(i), l) = 1$ if-and-only-if $array(i) = l$
  - Assume: the input is given in binary representation
    - Encryption: bit-by-bit
    - $a, b \in \{0,1\}^l$: the patterns whose equality we wish to determine
  - $isEqual_t(a, b) = \prod_{j\in[t]} \left(1 - \left(a_j - b_j\right)^2\right) \ mod\ p$