WIDESKIES: SCALABLE PRIVATE INFORMATION RETRIEVAL

ELLISON ANNE WILLIAMS AND FRIENDS

Abstract. We present Wideskies, an algorithm for private information search and retrieval. The algorithm operates in both streaming and batch contexts and is scalable within a distributed cloud environment.

1. Introduction

In this paper we present Wideskies, a private stream search algorithm that combines the filtering technique of [3] with the "slicing" technique of [1] to obtain an algorithm that is very bandwidth-efficient and protects the confidentiality of the search terms and results. Leveraging the Paillier homomorphic cryptosystem, this private information search and retrieval algorithm operates in both batch and streaming contexts and is scalable within an distributed cloud environment.

The paper is organized as follows: the variant of Paillier encryption used in Wideskies is discussed in Section 2 and Section 3 presents Wideskies in its streaming, batch, standalone, and distributed variants.

2. Paillier encryption

The Paillier encryption system was introduced by Pascal Paillier in [4]. In that paper Paillier proves that the system is semantically secure if and only if the decisional composite residuosity problem[1] is intractable, and that this problem is no harder than factoring. It is a standard result (see for example Theorem 11.6 of [2]) that semantic security of a public-key encryption scheme is enough to guarantee indistinguishability for multiple encryptions. This is critical for the security of Wideskies.

An instantiation of the Paillier cryptosystem is based on an underlying modulus $N = pq$ where p and q are primes of roughly the same length. The plaintext space is the (additive) group $\mathbb{Z}/N\mathbb{Z}$ and the ciphertext space is the (multiplicative) group $(\mathbb{Z}/N^2\mathbb{Z})^*$.

[1]Let $n \in \mathbb{N}$ be composite and let $z \in \left(\mathbb{Z}/n^2\mathbb{Z}\right)^*$. The decisional composite residuosity problem is to determine if z is an n^{th} root modulo n^2.

The public key is the modulus N and the private key is the factorization $N = pq$. We denote a Paillier key pair with modulus $N = pq$ by an ordered pair

$$(\{p, q\}, N)$$

where the first component is the private key and the second component is the public key.

The Paillier encryption and decryption procedures are presented in Algorithm 1 and Algorithm 2. As part of the Paillier decryption process we will require the *Carmichael λ function*, which gives the exponent of the multiplicative group of units modulo n. In general for $0 < n \in \mathbb{N}$ with prime factorization $n = p_1^{e_1} \dots p_k^{e_k}$,

$$\lambda(n) = \operatorname{lcm}(\{\lambda(p_i^{e_i}) : i = 1, \dots, k)\}).$$

Notice that for an odd prime p and $0 < e \in \mathbb{N}$

$$\lambda(p^e) = p^{e-1}(p-1) = \varphi(p^e)$$

so in this case Carmichael's λ and Euler's φ agree. In addition, for a Paillier modulus $N = pq$ we have

$$\lambda(N) = \operatorname{lcm}(p-1, q-1)$$

which is coprime to N and so has a multiplicative inverse modulo N.

As in [4], for $S_n = \{u < N^2 \,:\, u = 1 \bmod N\}$, let L be the function:

$$L(u) = \frac{u-1}{N}$$

which is well defined over $(\mathbb{Z}/N^2\mathbb{Z})^*$.

Algorithm 1 Paillier encryption and decryption.

procedure PAILLIER ENCRYPTION
  given the modulus N, an element $g \in (\mathbb{Z}/N^2\mathbb{Z})^*$ whose order is a nonzero multiple of N, and a message $m \in \mathbb{Z}/N\mathbb{Z}$
  select a random value $\zeta \in (\mathbb{Z}/N\mathbb{Z})^*$
  return $\mathcal{E}(m) = g^m \zeta^N \bmod N^2$

procedure PAILLIER DECRYPTION
  given the modulus and its factorization $N = pq$ and a ciphertext $c \in (\mathbb{Z}/N^2\mathbb{Z})^*$
  return $\mathrm{m} = \frac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)} \bmod N$

For Wideskies, set $g \in \mathbb{Z}/N^2\mathbb{Z}^* : g = (1+N)$; it is easily seen that g has order N in $\mathbb{Z}/N^2\mathbb{Z}^*$. Note that by the Bionomial Theorem:

$$\begin{aligned} g^m \bmod N^2 &= (1+N)^m \bmod N^2 \\ &= \sum_{k=0}^{m} \binom{m}{k} N^k \bmod N^2 \\ &= (1 + mN + \binom{m}{2} N^2 + ...) \bmod N^2 \\ &= (1+mN) \bmod N^2 \end{aligned}$$

For $g = (1+N)$, note that

$$\begin{aligned} L(g^{\lambda(N)} \bmod N^2) &= L((1+N)^{\lambda(N)} \bmod N^2) \\ &= L((1+\lambda(N)N) \bmod N^2) \\ &= \frac{(1+\lambda(N)N) \bmod N^2 - 1}{N} \\ &= \lambda(N) \end{aligned}$$

Then, for Wideskies, Paillier encryption and decryption is given by Algorithm 2.

Algorithm 2 Paillier encryption and decryption, $g = (1+N)$.

1: **procedure** PAILLIER ENCRYPTION
2: **given** the modulus N and a message $m \in \mathbb{Z}/N\mathbb{Z}$
3: **select** a random value $\zeta \in (\mathbb{Z}/N\mathbb{Z})^*$
4: **return** $\mathcal{E}(m) = (1+mN)\zeta^N \bmod N^2$

1: **procedure** PAILLIER DECRYPTION
2: **given** the modulus and its factorization $N = pq$ and a ciphertext $c \in (\mathbb{Z}/N^2\mathbb{Z})^*$
3: **set** $\mu = \lambda(N)^{-1} \bmod N$ ▷ Recall $\gcd(\lambda(N), N) = 1$
4: **set** $\hat{c} = c^{\lambda(N)} \bmod N^2$
5: **set** $\hat{m} = \frac{\hat{c}-1}{N} = L(c^{\lambda(N)} \bmod N^2)$ ▷ Division over $\mathbb{Z}$
6: **return** $\hat{m}\mu \bmod N$

If c is a Paillier ciphertext in Algorithm 2, then

$$N \mid (c^{\lambda(N)} - 1).$$

This ensures that the integer division on step 5 of the Paillier decryption procedure is valid.

Observe that

$$y \equiv (1+N)^x \bmod N^2 \rightarrow \frac{y-1}{N} \equiv x \bmod N$$

Then, for Algorithm 2, it is straightforward to verify the decryption:

$$\begin{aligned}\hat{m}\mu \bmod N &= L(((1+mN)\zeta^N)^{\lambda(N)} \bmod N^2) * (\lambda(N)^{-1} \bmod N) \\ &= L((1+mN)^{\lambda(N)} \bmod N^2) * (\lambda(N)^{-1} \bmod N) \\ &= L((1+N)^{m\lambda(N)} \bmod N^2) * (\lambda(N)^{-1} \bmod N) \\ &= (m\lambda(N) \bmod N) * (\lambda(N)^{-1} \bmod N) \\ &= m\end{aligned}$$

Let $\mathcal{E}(m)$ and $\mathcal{E}(m')$ be encryptions of the plaintexts $m, m' \in \mathbb{Z}_N$ with the same key N. Then, by the homomorphic properties of Paillier encryption, $\mathcal{E}(m)\mathcal{E}(m')$ is a valid encryption of $(m+m') \bmod N$:

$$D(\mathcal{E}(m)\mathcal{E}(m') \bmod N^2) = (m+m') \bmod N$$
$$D(\mathcal{E}(m)^k \bmod N^2) = km \bmod N,\ k \in \mathbb{N}$$

3. Wideskies

We now present the full Wideskies algorithm in detail.

We begin by describing Wideskies entirely in terms of the plaintext space and then present Wideskies in ciphertext space. Once we understand the desired operations on the plaintext space, it is straightforward to determine what operations must be carried out on the ciphertext in order to achieve the appropriate operations on the plaintext.

After describing the base algorithms of Wideskies, we will present the distributed variant.

3.1. Parameters.

In order to instantiate Wideskies, we must specify

- H – a keyed hash function with output length ℓ (which has range $[0, 2^l - 1]$); and
- N – a Paillier modulus of bit-length B.

Let H_k denote the keyed hash function H with key k.

In addition to H and N (and the values for ℓ and B that they provide) there are four other parameters that must be specified.

- τ – the number of targeted selectors (query terms);
- δ – the number of bits of data that are returned for each hit on a targeted selector;
- b – a "chunk size", in bits, that determines how the data for a hit are split amongst multiple plaintexts; and
- r – the number of plaintexts that can be returned per query period per targeted selector.

The requirements for these parameters are as follows:

(1) $2^{b\tau} < N$. There must be space in the modulus (which must be odd) to hold all the data even if every targeted selector hits with the maximum possible value. This also implies that $\tau < B/b$. Specifically, if B' is the highest bit position set in N, then $\tau \leq \lfloor \frac{B'}{b} \rfloor$.
(2) $\tau \leq 2^{\ell}$. For efficiency reasons, there should be very few collisions between targeted selectors under H; in some cases, it may be required that there be no collisions. In addition, τ should be significantly smaller than 2^{ℓ} in order to limit the number of false hits.
(3) $b \mid \delta$. The data returned for a hit must split evenly into b-bit chunks.
(4) $(\delta/b) \mid r$. The number of plaintexts returned per query period must be a multiple of the number of chunks for each hit. If this is not true, then there are always plaintexts that carry no data and so are wasted.
(5) The hash function H should be pseudorandom, but it does not need to be a cryptographically secure hash. As we'll see in Section 3.2.1, it is necessary that the hash function for the chosen key has no collisions amongst the targeted selectors unless the response data include additional indicators of the selector.

Note that H, ℓ, N, B, δ, b, and r are all specified in the clear. As we've already indicated, this gives a bound on τ since $2^{b\tau} < N$. If τ is sensitive, then the other parameters may be modified (in particular by making ℓ and N unnecessarily large) but at a cost in performance.

3.2. **Plaintext version.** With appropriate parameters selected we can now describe Wideskies as it operates on the plaintext space.

3.2.1. *Query formation.* The main query formation algorithm is presented in Algorithm 3. In some circumstances, a slightly different query formation algorithm, Algorithm 4, may be used. Keep in mind that we are describing plaintext actions here; in Section 3.3 we'll translate these tasks to their ciphertext-space equivalents.

Let $T_0, \ldots, T_{\tau-1}$ be the list of targeted selectors.

Notice that in step 6 there is at most one value $j \in \{0, \ldots, \tau - 1\}$ with $i = H_k(T_j)$ since there are no collisions amongst hashes of targeted values, so the E_i are well-defined.

The query consists of the E_i values, the hash function H, the key k, and the modulus N.

It may be important that the targets not collide under H_k because H will tend to be a short hash, and so if two targeted selectors collide

Algorithm 3 Query Formation Algorithm version 1

1: Choose a random key k for H.
2: Compute $H_k(T_0), \ldots, H_k(T_{\tau-1})$.
3: **while** card $(\{H_k(T_0), \ldots, H_k(T_{\tau-1})\}) < \tau$ **do**
4: **go to** 1 ▷ If there are hash collisions, pick a new key.
5: **for** $i = 0, \ldots, 2^\ell - 1$ **do**
6: Set

$$E_i = \begin{cases} 2^{jb} & \text{if } i = H_k(T_j); \\ 0 & \text{otherwise.} \end{cases}$$

7: **return** $\{E_0, \ldots, E_{2^\ell-1}, H, k, N\}$

it may not be possible to determine which data correspond to each selector. However, if the response data include additional indicators of the selector (for example, a longer hash of the actual selector), then Algorithm 4 can be used which does not require there be no collisions amongst hash values of the targeted selectors.

Algorithm 4 Query Formation Algorithm version 2

1: Choose a random key k for H.
2: Compute $H_k(T_0), \ldots, H_k(T_{\tau-1})$.
3: **for** $i = 0, \ldots, 2^\ell - 1$ **do**
4: Set $S_i = \{r \in \{0, \ldots, \tau - 1\} \mid H_k(T_r) = i\}$
5: **if** $S_i = \emptyset$ **then**
6: Set $E_i = 0$
7: **else**
8: Set $E_i = 2^{jb}$ where $j = \min(S_i)$
9: **return** $\{E_0, \ldots, E_{2^\ell-1}, H, k, N\}$

If collisions are allowed in query formation they must be disambiguated using the data supplied in the response.[2]

Notice that in both Algorithms 3 and 4 we have included the Paillier modulus N with the query. In the plaintext version this is not actually necessary, since in steps 6 and 7 of Algorithm 5 no modular reduction actually takes place. We have included N here because it will be required in the ciphertext version (so that the responder can compute

[2]If Wideskies is used with many similar selector lists but different hash functions (or keys), then it is better to allow collisions amongst the targeted selectors since forbidding collisions may leak data about the targeted selectors.

N^2), and our goal is to have the ciphertext version be an exact duplicate of the plaintext version except that it is performed under Paillier encryption.

3.2.2. *Response initialization.* At the outset the responder must initialize $2^\ell + r$ numbers. These are:

(1) 2^ℓ counters, called $c_0, \ldots, c_{2^\ell - 1}$, all starting at 0; and
(2) r Paillier plaintext values, $Y_0, \ldots, Y_{r-1}$, all initialized to 0.

3.2.3. *Handling the stream.* We view data coming in to the responder as arriving as ordered pairs $\mathbf{T} = \{(T, D)\}$ where T is the value on which targets are identified, and D is the data that should be returned if T corresponds to a targeted selector. Note that T is the result of a function applied to D where that function could be as simple as keyword extraction or something far more complex.

When a pair (T, D) arrives, the responder processes it as described in Algorithm 5.

Each data element D is interpreted as a bit-string, with each b-bit chunk D_i interpreted as a big-endian unsigned integer in step 9, with D_0 coming from the lowest-addressed b bits of D. For example, if, as a bit-string, $D = 011010$ and $b = 3$, then the resulting chunks are

$$D_0 = 3 \to 011$$
$$D_1 = 2 \to 010$$

At the end of the query period, the responder sends the Y_i values back to the querier.

Algorithm 5 Stream processing, plaintext version

1: **Input**: $\mathbf{T} = \{(T, D)\}$
2: **for** $(T, D) \in \mathbf{T}$ **do**
3: Compute $\mathcal{T} = H_k(T)$
4: **if** $c_\mathcal{T} + \frac{\delta}{b} > r$ **then** ▷ The space allocated for target T is full.
5: **return**
6: **else**
7: Split D into b-bit chunks $D_0 \| D_1 \| \ldots \| D_{(\delta/b)-1}$.
8: **for** $i = 0, \ldots, (\delta/b) - 1$ **do**
9: Set $\mathcal{D}_i = D_i E_\mathcal{T} \bmod N$ ▷ Nonzero only if $E_\mathcal{T} \neq 0$.
10: Set $Y_{i+c_\mathcal{T}} = Y_{i+c_\mathcal{T}} + \mathcal{D}_i \bmod N$
11: Set $c_\mathcal{T} = c_\mathcal{T} + (\delta/b)$
12: **Output**: $Y_0, \ldots, Y_{r-1}$

Notice that if $E_{\mathcal{T}} = 0$, then none of the Y_i values are changed. If $E_{\mathcal{T}} \neq 0$ then this operation places the bits of D_i into bit positions $jb, \ldots, (j+1)b-1$ of $Y_{i+c_{\mathcal{T}}}$ where $\mathcal{T} = H_k(T_j)$. Moreover, since we increment the counter $c_{\mathcal{T}}$, we will never attempt to place more bits into these positions and so the data we have placed there will be successfully returned.

It is also important to recognize that if a single target is seen too many times then some data will be lost. This is what is happening in steps 4-5. Careful selection of the parameters (especially δ and r), along with an understanding of the number of records (both targeted and non-targeted), in the stream during a query period can make the probability that data loss is as small as desired.

3.2.4. *Data recovery.* Recovering the data is easy for the querier. First, write each Y_i value in base-2^b as

$$Y_i = \sum_{j=0}^{\tau-1} 2^{jb} P_j.$$

Each P_j is a non-negative b-bit integer and so Y_i is strictly less than N. The b-bits of P_j may contain data related to target T_j, and will always be 0 if target T_j was not seen enough times to have required the use of Y_i. Thus, the data recorded for the first hit on target T_0 are the concatenation of the low b-bits of $Y_0, \ldots, Y_{(\delta/b)-1}$. Similarly, the data recorded for the first hit on target T_j are the concatenation of the bits at positions $jb, \ldots, (j+1)b-1$ of $Y_0, \ldots, Y_{(\delta/b)-1}$. In general, the data recorded for the η^{th} hit on target T_j is the concatenation of the bits at positions $jb, \ldots, (j+1)b-1$ of $Y_{(\eta-1)\delta/b}, \ldots, Y_{(\eta\delta/b)-1}$.

Slightly more formally, Algorithm 6 describes how to recover all data recorded for target T_j. Note that M, the $Y_{(\eta-1)(\delta/b)+i}$, and the D_i are all interpreted as big-endian unsigned integers.

Algorithm 6 Data recovery, plaintext version

1: Set $M = 2^{jb}(2^b - 1)$.
2: **for** $\eta = 1, \ldots, (rb/\delta)$ **do** ▷ At most rb/δ hits can be returned.
3: **for** $i = 0, \ldots, (\delta/b) - 1$ **do** ▷ Each hit uses δ/b chunks.
4: Set $D_i = Y_{(\eta-1)(\delta/b)+i} \& M$ ▷ "&" denotes bit-wise `AND`.
5: Set $D_i = D_i / 2^{jb}$ ▷ Step 4 ensures $2^{jb} \mid D_i$
6: Set $X_\eta = D_0 \| D_1 \| \ldots \| D_{(\delta/b)-1}$
7: **return** $X_1, \ldots, X_{(rb/\delta)}$ ▷ the data corresponding to selector T_j

Notice that the bit-wise `AND` in step 4 has the effect of setting all bits of D_i to 0 except for those in positions $jb, \ldots, (j+1)b-1$, which are exactly the positions that hold data relevant to target T_j. Then the division in step 5 moves those b bits into positions $0, \ldots, b-1$, resulting in D_i being a number between 0 and $2^b - 1$, and so interpretable as a b-bit number. Concatenating the bit-strings corresponding to these b-bit numbers results in recovery of the original data split into b-bit chunks by the responder.

3.3. **Ciphertext version.** Now that we understand what should happen in the plaintext space, it is straightforward to transform this system into one that uses Paillier encryption to achieve private stream search. Essentially, all that we do is encrypt the query values $E_0, \ldots, E_{2^\ell-1}$ and then modify the responder's portion of the algorithm to use the homomorphic properties of Paillier encryption to manipulate the ciphertext in a way that produces the same plaintext operations as we used in Section 3.2. Then, when the query is returned, the querier will decrypt the returned values and recover the original data using the same technique as with the plaintext version of the system.

The system parameters for the real Widesкies are the same as for the plaintext version. Throughout this section, we have a fixed Paillier key pair, $(\{p, q\}, N)$, and let $\mathcal{E}(x)$ denote an encryption of x using that key pair.

3.3.1. *Query formation.* Let $T_0, \ldots, T_{\tau-1}$ be the list of targeted selectors. Algorithm 7 is the ciphertext version of Algorithm 3, and Algorithm 8 is the ciphertext version of Algorithm 4.

Algorithm 7 Query formation, ciphertext version 1

1: Choose a random key k for H.
2: Compute $H_k(T_0), \ldots, H_k(T_{\tau-1})$.
3: **while** card $(\{H_k(T_0), \ldots, H_k(T_{\tau-1})\}) < \tau$ **do**
4: **go to** 1
5: **for** $i = 0, \ldots, 2^\ell - 1$ **do**
6: Set

$$\mathcal{E}_i = \begin{cases} \mathcal{E}(2^{jb}) & \text{if } i = H_k(T_j) \text{ for some } j \in \{0, \ldots, \tau-1\} \\ \mathcal{E}(0) & \text{otherwise.} \end{cases}$$

7: **return** $\{\mathcal{E}_0, \ldots, \mathcal{E}_{2^\ell-1}, H, k, N\}$

As in Section 3.2.1, if we wish to relax the requirement that there be no hash collisions amongst selectors we can instead do Algorithm 8.

Algorithm 8 Query formation, ciphertext version 2

Choose a random key k for H.
Compute $H_k(T_0), \ldots, H_k(T_{\tau-1})$.
for $i = 0, \ldots, 2^\ell - 1$ **do**
    Set $S_i = \{r \in \{0, \ldots, \tau - 1\} \mid H_k(T_r) = i\}$
    if $S_i = \emptyset$ **then**
        Set $\mathcal{E}_i$ to an encryption of 0
    else
        Set $\mathcal{E}_i$ to an encryption of 2^{jb} where $j = \min(S_i)$
return $\{\mathcal{E}_0, \ldots, \mathcal{E}_{2^\ell-1}, H, k, N\}$

3.3.2. *Response initialization.* Similar to the plaintext version, the responder must initialize $2^\ell + r$ values. These are:

(1) 2^ℓ counters, called $c_0, \ldots, c_{2^\ell-1}$, all starting at 0; and
(2) r Paillier ciphertext values $\mathcal{Y}_0, \ldots, \mathcal{Y}_{r-1}$, all initialized to 1.

It is a feature of Paillier encryption that 1 is always a valid ciphertext value for the plaintext value 0, so we have initialized the $\mathcal{Y}_i$'s to be encryptions of 0.

3.3.3. *Handling the stream.* We maintain the view that data arrives to the responder as ordered pairs $\mathbf{T} = \{(T, D)\}$ where T is the value on which targets are identified, and D is the data that should be returned if T corresponds to a targeted selector.

When a pair (T, D) arrives, the responder processes it following Algorithm 9.

Just as with the plaintext version, a data element D is interpreted as a bit-string with D_0 occupying the low-order b-bits and with each b-bit chunk D_i interpreted as a big-endian unsigned integer.

At the end of the query period the responder sends the $\mathcal{Y}_i$ values back to the querier.

3.3.4. *Data Recovery.* Recovering the data is nearly identical to the plaintext version given in Section 3.2.4. In this case we start with ciphertext values $\mathcal{Y}_i$, but the querier can decrypt these to obtain plaintext values Y_i. These are processed identically to the plaintext version to reveal the returned data.

If hash collisions were allowed amongst the targeted selectors, data recovery must include an additional step of disambiguating the colliding targeted selectors.

In practice, since hash collisions may occur between targeted and non-targeted selectors, data recovery should include an additional data

Algorithm 9 Stream processing, ciphertext version

1: **Input**: $\mathbf{T} = \{(T, D)\}$
2: **Initialize:**
3: Counters $c_i = 0\,,\ 0 \leq i \leq (2^l - 1)$
4: Paillier ciphertext values $\mathcal{Y}_j = 1\,,\ 0 \leq j \leq (r - 1)$
5: **for** $(T, D) \in \mathbf{T}$ **do**
6: Compute $\mathcal{T} = H_k(T)$
7: **if** $c_{\mathcal{T}} + \frac{\delta}{b} > r$ **then**
8: **return**
9: **else**
10: Split D into b-bit chunks, $D = D_0 \| D_1 \| \ldots \| D_{(\delta/b)-1}$
11: **for** $i = 0, \ldots, (\delta/b) - 1$ **do**
12: Set $\mathcal{D}_i = \mathcal{E}_{\mathcal{T}}^{D_i} \bmod N^2$
13: Set $\mathcal{Y}_{i+c_{\mathcal{T}}} = \mathcal{Y}_{i+c_{\mathcal{T}}} \mathcal{D}_i \bmod N^2$
14: Set $c_{\mathcal{T}} = c_{\mathcal{T}} + (\delta/b)$
15: **Output**: $\mathcal{Y}_0, \ldots, \mathcal{Y}_{r-1}$

disambiguation step to remove false positives. Depending on the manner in which data disambiguation is acheived, the false positive rate may be reduced to zero. For example, if the selector T is embedded/inserted into D in step 10 of Algorithm 9, the returned data may be checked by the querier to ensure that the embedded selector matches the corresponding target selector. If not, the returned data element is discarded. In this manner, all false positive results are removed. Alternatively, a secondary hash of the selector T may be embedded instead of the selector itself. Due to the possibility of secondary hash collisions with the target selector, this method will calculably reduce false positives but not eliminate them entirely.

3.3.5. *Minimizing Response Space.* In the interest of minimizing the response size it is possible to only return $\mathcal{Y}_i$ values that may contain encrypted data. Since the responder is maintaining the counters, she is aware of any $\mathcal{Y}_i$ value that was never modified during stream processing, and so can simply omit those values from the response.

3.4. **Distributed Version.** The distributed version of Wideskies may be used in both streaming and batch computing contexts.

For the distributed case, we view the encrypted query as being performed via the construction and manipulation of a matrix $M = (m_{i,j})$ where $0 \leq i \leq (2^l - 1)$ and $0 \leq j \leq (r - 1)$.

The matrix equivalent of Algorithm 9 is given in Algorithm 10.

Algorithm 10 Responder - Matrix Variant

Input: $\mathbf{T} = \{(T, D)\}$
Initialize:
    Counters $c_i = 0\,,\ 0 \leq i \leq (2^l - 1)$
    Paillier ciphertext values $\mathcal{Y}_j = 1\,,\ 0 \leq j \leq (r-1)$
for $(T, D) \in \mathbf{T}$ **do**
    Compute $\mathcal{T} = H_k(T)$ ▷ View as the row index of $M : m_{\mathcal{T},j}$
    if $c_{\mathcal{T}} + \frac{\delta}{b} > r$ **then**
        return
    else
        Split D into b-bit chunks, $D = D_0 \| D_1 \| \ldots \| D_{(\delta/b)-1}$
        for $k = 0, \ldots, (\delta/b) - 1$ **do**
            Set $m_{\mathcal{T}, c_{\mathcal{T}}+k} = \mathcal{E}_{\mathcal{T}}^{D_k} \bmod N^2$
        Set $c_{\mathcal{T}} = c_{\mathcal{T}} + (\delta/b)$
for $0 \leq j \leq (r-1)$ **do**:
        $\mathcal{Y}_j = \prod_{i=0}^{2^l-1} m_{i,j}$
Output: $\mathcal{Y}_0, \ldots, \mathcal{Y}_{r-1}$

Using the matrix variant, Wideskies may be computed in a distributed environment as in Algorithm 11.

At the conclusion of Algorithm 11 the $\mathcal{Y}_0, \ldots, \mathcal{Y}_{r-1}$ column product values are then returned to the querier where they can be decrypted and the data can be recovered.

Algorithm 11 Responder - Distributed Variant

Input: $\mathbf{T} = \{(T, D)\}$
for $(T, D) \in \mathbf{T}$ **do** in parallel
    Compute $\mathcal{T} = H_k(T)$ ▷ View as the row index of $M : m_{\mathcal{T},j}$
    Split D into b-bit chunks, $D = D_0 \| D_1 \| \ldots \| D_{(\delta/b)-1}$
    Form $\mathbf{D} = \{D_k : 0 \leq k \leq (\delta/b) - 1\}$
    Emit $(\mathcal{T}, \mathbf{D})$
for each $\mathcal{T}$ **do** in parallel
    Initialize $c_{\mathcal{T}} = 0$
    while $c_{\mathcal{T}} < r$ **do**
        for each $(\mathcal{T}, \mathbf{D})$ **do**
            for each $D_k \in \mathbf{D}$, $0 \leq k \leq \ldots, (\delta/b) - 1$ **do**
                Set $m_{\mathcal{T},c_{\mathcal{T}}} = \mathcal{E}_{\mathcal{T}}^{D_k} \bmod N^2$
                Emit $(c_{\mathcal{T}}, m_{\mathcal{T},c_{\mathcal{T}}})$
                $c_{\mathcal{T}} = c_{\mathcal{T}} + 1$
for $0 \leq j \leq (r-1)$ in parallel **do**:
            $\mathcal{Y}_j = \prod_{i=0}^{2^l-1} m_{i,j}$
Output: $\mathcal{Y}_0, \ldots, \mathcal{Y}_{r-1}$

References

[1] Tingjian Ge and Stan Zdonik. Answering aggregation queries in a secure system model. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 519–530. VLDB Endowment, 2007.

[2] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2nd edition, 2014.

[3] Rafail Ostrovsky and III Skeith, William E. Private searching on streaming data. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 223–240. Springer Berlin Heidelberg, 2005.

[4] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT '99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

National Security Agency, 9800 Savage Rd., Fort George G. Meade, MD 20755, USA