APACHE PIRK

Mathematics & Algorithms

Walter Ray-Dulany

raydulany@apache.org

Introduction

Pirk's Wideskies Algorithm

Pirk uses the Wideskies algorithm to accomplish scalable PIR.

This algorithm can be broken down into two distinct conceptual pieces:

- Paillier Encryption
- The Query-Response-Result algorithms

Before we begin those however, we take a (happily brief) diversion into the language of the mathematics involved in this deck.

Language Preliminaries

Language Preliminaries

The Paillier scheme employs a small amount of group theoretic notation. Let's go over that notation briefly.

Language Preliminaries

- $\mathbf{Z}/N\mathbf{Z}$: This is the group of integers modulo N; it can be thought of as all numbers $0 \leq k < N$, with modular addition (e.g. for $N = 5$, $1 + 7 \equiv 3 \mod N$).

  This is a group under addition.

- $(\mathbf{Z}/N\mathbf{Z})^\times$: This is the multiplicative group of integers modulo N, also called the units of $\mathbf{Z}/N\mathbf{Z}$. Sometimes denoted $(\mathbf{Z}/N\mathbf{Z})^*$, this is the set of $0 \leq k < N$ that are relatively prime to N (that is, k and N share no factors, or equivalently the greatest common denominator (gcd) of k and N is 1). One can also think of this as the set of $k \in \mathbf{Z}/N\mathbf{Z}$ such that there exists a $k^{-1} \in \mathbf{Z}/N\mathbf{Z}$ with $k \cdot k^{-1} \equiv 1 \mod N$.

  This is a group under multiplication.

Language Preliminaries

Using the above notation, we can see that

$$\left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times = \{0 \leq k < N^2 : \gcd(k, N^2) = 1\}.$$

If N happens to be an RSA modulus, $N = pq$, p and q primes, then $\left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$ is just all numbers between 0 (inclusive) and N^2 (exclusive) that are not divisible by either p or q.

Language Preliminaries

- Order In $\mathbf{Z}/N\mathbf{Z}$: The order of an element $k \in \mathbf{Z}/N\mathbf{Z}$ is the least integer e such that $e \cdot k = 0 \mod N$.
- Order in $\left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$: The order of an element $a \in \left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$ is the least integer e such that $a^e = 1 \mod N^2$.

In both cases, order is well defined (i.e. it exists and makes sense) for all elements of the groups.

Language Preliminaries

For a more in depth discussion of these, and closely related, terms, please see

`http://www.math.nagoya-u.ac.jp/~richard/teaching/s2015/Group_2.pdf`

Paillier Encryption

Paillier Encryption

Paillier encryption is a partially homomorphic public key scheme that relies on the function

$$\mathcal{E}_g : \mathbf{Z}/N\mathbf{Z} \times (\mathbf{Z}/N\mathbf{Z})^\times \to \left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$$

given by

$$\mathcal{E}_g(x, y) = g^x y^N \mod N^2,$$

$g \in \left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$. Here, $\mathbf{Z}/N\mathbf{Z}$ is the plaintext space and $\left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$ is the ciphertext space.

When the order of g is a non-zero multiple of N, $\mathcal{E}_g$ is a bijection.

Paillier Prerequisites

- Public key: (N, g), N an RSA modulus $N = pq$, p and q primes of approximately the same bit-length, and $g \in \left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$ such that the order of g is a nonzero multiple of N.
- Private key: $\lambda(N)$, where λ is the Carmichael function

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

  that gives the exponent of $(\mathbf{Z}/N\mathbf{Z})^\times$.
- Plaintext space: $\mathbf{Z}/N\mathbf{Z}$.
- Ciphertext space: $\left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$.

We can also consider the pair (p, q) to be the private key, as $\lambda(N)$ is quickly and easily derived from it. Note that $\lambda(N)$ is coprime to N.

What Do We Mean By ‘Homomorphic Encryption’?

An encryption scheme is fully homomorphic if it is a homomorphism from plaintext space to ciphertext space for arbitrary operations and arbitrary numbers of such operations. If this definition seems squishy and not very mathematical, that’s because it is; it’s hard to find a proper mathematical definition of this term.

An encryption scheme is partially homomorphic if it is a homomorphism for only some operations, or for only a few consecutive operations.

Paillier Encryption is Homomorphic

Paillier encryption is a partial homomorphism between addition in $\mathbf{Z}/N\mathbf{Z}$ and multiplication in $\left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$.

Denote Paillier encryption by $\mathcal{E}_g$ and decryption by $\mathcal{D}_g$, and let m and $m' \in \mathbf{Z}/N\mathbf{Z}$. Then

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

Note that the second equality follows immediately from the first.

General Paillier Algorithm

Paillier Supporting Function

Let $X = \{u < N^2 : u = 1 \mod N\}$ and let $L : X \to \mathbf{Z}/N\mathbf{Z}$ be defined by

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

This function is well defined over $\left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$.

General Paillier Encryption

The general Paillier algorithm differs only slightly from Pirk's version.

Algorithm 1 General Paillier encryption and decryption.

```
1: procedure Paillier encryption
2:     given $N$, a random $g \in \left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$ of order a nonzero
       multiple of $N$, and a message $m \in \mathbf{Z}/N\mathbf{Z}$
3:     select a random value $\zeta \in (\mathbf{Z}/N\mathbf{Z})^\times$
4:     return $\mathcal{E}(m) = g^m \zeta^N \bmod N^2$

1: procedure Paillier decryption
2:     given $N$, $\lambda(N)$, $g$, and ciphertext $c \in \left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$
3:     return m $= \frac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)} \bmod N$
```

Paillier Works

It is a straightforward exercise to check that

- $D(\mathcal{E}(m)) = m$
- $D(\mathcal{E}(m)\mathcal{E}(m') \bmod N^2) = (m + m') \bmod N$

Paillier As Used In Wideskies

Paillier As Used In Wideskies

The version of Paillier used in Wideskies is a computationally simpler variant of the full Paillier scheme that sacrifices no security over the general case.

Converting Between The Two

Pirk's simplified version of Paillier simply uses

$$\begin{aligned} g &\equiv 1 + N \mod N^2 \\ \implies L(g^{\lambda(N)} \mod N^2) &= \lambda(N), \end{aligned}$$

the proof of which is a straightforward exercise.

Paillier As Used In Wideskies

Algorithm 2 Paillier encryption and decryption

```
1: procedure Paillier encryption
2:     given $N$ and a message $m \in \mathbf{Z}/N\mathbf{Z}$
3:     select a random value $\zeta \in \left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$
4:     return $\mathcal{E}(m) = (1 + mN)\zeta^N \bmod N^2$
1: procedure Paillier decryption
2:     given $N$, $\lambda(N)$, and a ciphertext $c \in \left(\mathbf{Z}/N^2\mathbf{Z}\right)^\times$
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 $\hat{m} = L(c^{\lambda(N)} \bmod N^2)$
6:     return $\hat{m}\mu \bmod N$
```

Paillier Reference

For more on Paillier encryption and the (hypothesized) hard problem upon which it is based, see

`https://pirk.incubator.apache.org/papers/1999_asiacrypt_paillier_paper.pdf`

on Pirk's website.

Wideskies

Wideskies Parameters

The algorithm requires the following parameters, which are not independent (see the next slide).

- N, the Paillier modulus
- B, the bit-length of N
- H (or H_k), a keyed hash function (with key k)
- ℓ, the bit length of the output of H, i.e. $H_k : \mathbf{Z} \to (\mathbf{Z}/2\mathbf{Z})^\ell$
- τ, the number of search terms
- δ, the number of bits of data returned for each search hit
- b the chunk size, in bits, determining how data is split among responses.
- r, the number of responses that can be returned per query request period per search term

Parameter Relationships

- $2^{b\tau} < N$: there must be space in the modulus to hold all the data, even if each search term hits as often as possible.
- $\tau < 2^{\ell}$: Although the paper permits search term hash collisions, Pirk does not permit them. Typically $\tau \ll 2^{\ell}$
- $b|\delta$: Chunk size must evenly divide the data size
- $\frac{\delta}{b}|r$: the number of chunks per returned datum must divide the number of responses, for bandwidth efficiency.
- H: Must be pseudo-random but need not be cryptographically secure.

Public Parameters

All of

$$H, \ell, N, B, \delta, b, \text{and } r$$

are public, that is, must be shared between the client and server.

Note that the fact that $2^{b\tau} < N$ gives some information on the number of search terms the client is using; the amount of this information can be decreased without bound by choosing N and ℓ to be much larger than would be otherwise necessary; this necessarily causes a performance hit.

Wideskies Algorithm, Without Encryption

Wideskies Without Encryption?

The Wideskies algorithm is of sufficient complexity that it can be useful to go through the algorithm without the encryption and decryption steps first, in order to orient ourselves.

After, it will be straightforward to see the changes that using the Paillier encryption requires.

Query, Without Encryption

The Query Algorithm, Without Encryption

Let $T_0, \ldots, T_{\tau-1} \in \mathbf{Z}/N\mathbf{Z}$ be our search terms.

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 $\mathrm{card}(\{H_k(T_0), \ldots, H_k(T_{\tau-1})\}) < \tau$ 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 $\{E_0, \ldots, E_{2^\ell-1}, H, k, N\}$
```

Query Notes, Without Encryption

Since $\tau \ll 2^\ell$, we expect most of the E_i to be zero.

We will typically denote $H_k(T)$ by $\mathcal{T}$ and its associated E by $E_{\mathcal{T}}$. If we wish to keep track of a specific T we will write $\mathcal{T}_j$ and $E_{\mathcal{T}_j}$.

Response, Without Encryption

Response Initialization, Without Encryption

We must initialize some values before forming the response.

1. $c_0, \ldots, c_{2^\ell-1} = 0$, counters to keep track of the number of times each $E_{\mathcal{T}}$ has been seen.
2. $Y_0, \ldots, Y_{r-1} = 0$, response vectors.

Response Data, Without Encryption

Responder information comes in pairs (T, D) where T is a (potential) search term, and D is T's associated response datum, which will be returned if T is a search term.

We view D as a δ-long bit stream $(d_0, \ldots, d_{\delta-1})$, and break D up into δ/b chunks D_i as

$$D_i = (d_{i \cdot b}, d_{i \cdot b+1}, \ldots, d_{(i+1) \cdot b-1}),\ i = 0, \ldots, \delta/b - 1.$$

For example, if $D = 011010$ and $b = 3$, then

$$D_0 = 011$$
$$D_1 = 010$$

Response Algorithm, Without Encryption

Algorithm 4 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 term $T$ is full.
5:         **return**
6:     **else**
7:         Split $D$ into $b$-bit chunks $D_0, \ldots, D_{(\delta/b)-1}$.
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**: $Y_0, \ldots, Y_{r-1}$
```

Response Example, Without Encryption

Let's look at how the response would look on the first four (T, D) pairs that pass through the algorithm.

Response Example Setup, Without Encryption

Suppose that among our search terms are T and T', with

$$H_k(T) = j \text{ and}$$
$$H_k(T') = j'.$$

Suppose that T'', with $H_k(T'') = j''$, is *not* a search term.

Let the responder see, in order, the pairs (T, D^0), (T', D^1), (T'', D^2), (T, D^3).

The Y_i are formed by summing down the columns in following matrices.

Response Example Start, Without Encryption

No terms have yet been evaluated.

Index	Y_0	$\cdots$	$Y_{\delta/b-1}$	$Y_{\delta/b}$	$\cdots$	$Y_{2\delta/b-1}$
$\vdots$						
j	0	$\cdots$	0	0	$\cdots$	0
$\vdots$						
j'	0	$\cdots$	0	0	$\cdots$	0
$\vdots$						
j''	0	$\cdots$	0	0	$\cdots$	0
$\vdots$						

$c_j = 0,\ c_{j'} = 0,\ c_{j''} = 0.$

Response Example: First Term, Without Encryption

(T, D^0) enters and is proccessed; hit:

Index	Y_0	$\cdots$	$Y_{\delta/b-1}$	$Y_{\delta/b}$	$\cdots$	$Y_{2\delta/b-1}$
$\vdots$						
j	$\boldsymbol{D_0^0} 2^{jb}$	$\cdots$	$\boldsymbol{D_{\delta/b-1}^0} 2^{jb}$	0	$\cdots$	0
$\vdots$						
j'	0	$\cdots$	0	0	$\cdots$	0
$\vdots$						
j''	0	$\cdots$	0	0	$\cdots$	0
$\vdots$						

$\boldsymbol{c_j = \delta/b}$, $c_{j'} = 0$, $c_{j''} = 0$.

Response Example Second Term, Without Encryption

(T', D^1) enters and is proccessed; hit:

Index	Y_0	$\cdots$	$Y_{\delta/b-1}$	$Y_{\delta/b}$	$\cdots$	$Y_{2\delta/b-1}$
$\vdots$						
j	$D^0_0 2^{jb}$	$\cdots$	$D^0_{\delta/b-1} 2^{jb}$	0	$\cdots$	0
$\vdots$						
j'	$\boldsymbol{D^1_0} 2^{j'b}$	$\cdots$	$\boldsymbol{D^1_{\delta/b-1}} 2^{j'b}$	0	$\cdots$	0
$\vdots$						
j''	0	$\cdots$	0	0	$\cdots$	0
$\vdots$						

$c_j = \delta/b$, $\boldsymbol{c_{j'} = \delta/b}$, $c_{j''} = 0$.

Response Example Third Term, Without Encryption

(T'', D^2) enters and is proccessed; no hit:

Index	Y_0	$\cdots$	$Y_{\delta/b-1}$	$Y_{\delta/b}$	$\cdots$	$Y_{2\delta/b-1}$
$\vdots$						
j	$D_0^0 2^{jb}$	$\cdots$	$D_{\delta/b-1}^0 2^{jb}$	0	$\cdots$	0
$\vdots$						
j'	$D_0^1 2^{j'b}$	$\cdots$	$D_{\delta/b-1}^1 2^{j'b}$	0	$\cdots$	0
$\vdots$						
j''	$\boldsymbol{D_0^2} \cdot 0$	$\cdots$	$\boldsymbol{D_{\delta/b-1}^2} \cdot 0$	0	$\cdots$	0
$\vdots$						

$c_j = \delta/b$, $c_{j'} = \delta/b$, $\boldsymbol{c_{j''} = \delta/b}$.

Response Example Fourth Term, Without Encryption

(T, D^3) enters and is proccessed; hit:

Index	Y_0	$\cdots$	$Y_{\delta/b-1}$	$Y_{\delta/b}$	$\cdots$	$Y_{2\delta/b-1}$
$\vdots$						
j	$D_0^0 2^{jb}$	$\cdots$	$D_{\delta/b-1}^0 2^{jb}$	$\boldsymbol{D_0^3} 2^{jb}$	$\cdots$	$\boldsymbol{D_{\delta/b-1}^3} 2^{jb}$
$\vdots$						
j'	$D_0^1 2^{j'b}$	$\cdots$	$D_{\delta/b-1}^1 2^{j'b}$	0	$\cdots$	0
$\vdots$						
j''	$\boldsymbol{D_0^2} \cdot 0$	$\cdots$	$\boldsymbol{D_{\delta/b-1}^2} \cdot 0$	0	$\cdots$	0
$\vdots$						

$\boldsymbol{c_j = 2\delta/b}$, $c_{j'} = \delta/b$, $c_{j''} = \delta/b$.

Result, Without Encryption

Result, Without Encryption

The algorithm for getting the results out of the response return is straightforward. To begin,

- Write $Y_i = \sum_{k=0}^{\tau-1} 2^{kb} P_{ki}$ in base 2^b, where P_{ki} is the value of the k^{th} row in the i^{th} column. Note each P_{ki} is b-bits long, and therefore $Y_i < N$.
- Y_i will have data on search term T if and only if T was seen $i + 1$ times before the responder returned.

Result Algorithm, Without Encryption

Algorithm 5 Data recovery, plaintext version

```
1: Set $M = 2^{jb}(2^b - 1)$                          ▷ $b$ 1s left-shifted $jb$ places.
2: for $\eta = 1, \ldots, (rb/\delta)$ do              ▷ At most $rb/\delta$ hits can be returned.
3:     for $i = 0, \ldots, (\delta/b) - 1$ do          ▷ Each hit uses $\delta/b$ 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 $T_j$
```

Wideskies Algorithm, With Encryption

Adding Encryption To The Mix

Adding encryption is straightforward. The following slides have the encryption-enabled algorithms, with the differences from the earlier slides in bold.

Query, Encrypted

Query, Encrypted

Algorithm 6 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 $\mathrm{card}(\{H_k(T_0), \ldots, H_k(T_{\tau-1})\}) < \tau$ 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 $\{\mathcal{E}_0, \ldots, \mathcal{E}_{2^\ell-1}, H, k, N\}$
```

Response, Encrypted

Response Initialization, Encrypted

As before, we must initialize some values before forming the response.

1. $c_0, \ldots, c_{2^\ell - 1} = 0$, counters to keep track of the number of times each $\mathcal{E}_\mathcal{T}$ has been seen.
2. **$\boldsymbol{Y_0, \ldots, Y_{r-1} = 1}$, response vectors.**

Response, Encrypted

Algorithm 7 Stream processing, ciphertext version

```
1:  **Input**: $\mathbf{T} = \{(T, D)\}$
2:  **Initialize:**
3:      Counters $c_i = 0\,,\ 0 \le i \le (2^l - 1)$
4:      Paillier ciphertext values $\mathcal{Y}_j = 1\,,\ 0 \le j \le (r-1)$
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, $D = D_0, \ldots, D_{(\delta/b)-1}$
11:         **for** $i = 0, \ldots, (\delta/b) - 1$ **do**
12:             **Set** $\boldsymbol{\mathcal{D}_i = \mathcal{E}_{\mathcal{T}}^{D_i}}$ **mod** $\boldsymbol{N^2}$
13:             **Set** $\boldsymbol{\mathcal{Y}_{i+c_{\mathcal{T}}} = \mathcal{Y}_{i+c_{\mathcal{T}}}\mathcal{D}_i}$ **mod** $\boldsymbol{N^2}$
14:         Set $c_{\mathcal{T}} = c_{\mathcal{T}} + (\delta/b)$
15: **Output**: $\mathcal{Y}_0, \ldots, \mathcal{Y}_{r-1}$
```

Result, Encrypted

Result, Encrypted (and then Decrypted)

Actually literally the same algorithm as before is used; the only difference is that we first decrypt the encrypted $\mathcal{Y}_i$.

Distributed Version

Distributed Version

The paper goes over how to do the distributed version; the change is straightforward, and our earlier example slides make it easy to see how it works.

Distributed Difference: Unencrypted Sums, Encrypted Products

Recall our example matrix:

Index	Y_0	$\cdots$	$Y_{\delta/b-1}$	$Y_{\delta/b}$	$\cdots$	$Y_{2\delta/b-1}$
$\vdots$						
j	$D_0^0 2^{jb}$	$\cdots$	$D_{\delta/b-1}^0 2^{jb}$	$D_0^3 2^{jb}$	$\cdots$	$D_{\delta/b-1}^3 2^{jb}$
$\vdots$						
j'	$D_0^1 2^{j'b}$	$\cdots$	$D_{\delta/b-1}^1 2^{j'b}$	0	$\cdots$	0
$\vdots$						
j''	$\boldsymbol{D_0^2} \cdot 0$	$\cdots$	$\boldsymbol{D_{\delta/b-1}^2} \cdot 0$	0	$\cdots$	0
$\vdots$						

Distributed Difference: Unencrypted Sums, Encrypted Products

When we moved to an encrypted algorithm, all of the D_i-long sums of E_i became $\mathcal{E}_i^{\mathcal{D}_i}$.

In the distributed version, we actually make matrix components rather than the fake matrix of D_i in certain bit-positions we had earlier.

In the matrix, rows are indexed by $0 \leq \mathcal{T} \leq 2^{\ell} - 1$, columns by $0 \leq j \leq r - 1$.

Distributed Difference: Unencrypted Sums, Encrypted Products

As before, let

$$\begin{aligned} H_k(T) &= j, \\ H_k(T') &= j', \text{ and} \\ H_k(T'') &= j'', \end{aligned}$$

with T and T' search terms and T'' not.

Notice that this time we won't simply discard the data D_2 from T''; we no longer multiply it by zero, but use it as the exponent of $\mathcal{E}_{j''}$, which is an encryption of 0.

Distributed Difference: Unencrypted Sums, Encrypted Products

The matrix in the encrypted setting:

Index	$\mathcal{Y}_0$	$\cdots$	$\mathcal{Y}_{\delta/b-1}$	$\mathcal{Y}_{\delta/b}$	$\cdots$	$\mathcal{Y}_{2\delta/b-1}$
$\vdots$						
j	$\mathcal{E}_j^{D_{\mathbf{0}}^{\mathbf{0}}}$	$\cdots$	$\mathcal{E}_j^{D^{\mathbf{0}}_{\delta/b-\mathbf{1}}}$	$\mathcal{E}_j^{D_{\mathbf{0}}^{\mathbf{3}}}$	$\cdots$	$\mathcal{E}_j^{D^{\mathbf{3}}_{\delta/b-\mathbf{1}}}$
$\vdots$						
j'	$\mathcal{E}_{j'}^{D_{\mathbf{0}}^{\mathbf{1}}}$	$\cdots$	$\mathcal{E}_{j'}^{D^{\mathbf{1}}_{\delta/b-\mathbf{1}}}$	1	$\cdots$	1
$\vdots$						
j''	$\mathcal{E}_{j''}^{D_{\mathbf{0}}^{\mathbf{2}}}$	$\cdots$	$\mathcal{E}_{j''}^{D^{\mathbf{2}}_{\delta/b-\mathbf{1}}}$	1	$\cdots$	1
$\vdots$						

Algorithm In Matrix Form

Algorithm 8 Responder - Matrix Variant

```
1:  **Input:** $\mathbf{T} = \{(T, D)\}$
2:  **Initialize:**
3:      Counters $c_i = 0\,,\ 0 \le i \le (2^l - 1)$
4:      Paillier ciphertext values $\mathcal{Y}_j = 1\,,\ 0 \le j \le (r-1)$
5:  for $(T, D) \in \mathbf{T}$ do
6:      Compute $\mathcal{T} = H_k(T)$                ▷ View as the row index of $M : m_{\mathcal{T},j}$
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 $k = 0, \ldots, (\delta/b) - 1$ do
12:             Set $m_{\mathcal{T},\, c_{\mathcal{T}}+k} = \mathcal{E}_{\mathcal{T}}^{D_k} \bmod N^2$
13:         Set $c_{\mathcal{T}} = c_{\mathcal{T}} + (\delta/b)$
14: for $0 \le j \le (r-1)$: do
15:             $\mathcal{Y}_j = \prod_{i=0}^{2^l-1} m_{i,j}$
16: **Output**: $\mathcal{Y}_0, \ldots, \mathcal{Y}_{r-1}$
```

Distributed Algorithm

Algorithm 9 Responder - Distributed Variant

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

‘Actual’ Example

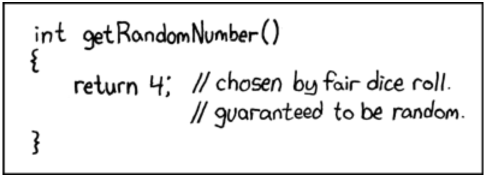
Actual Example Setup

We run through the above with actual numbers.

Let

- $N = 35$, $p = 5$, $q = 7$, $\lambda(N) = 12$, $B = 5$.
- τ, the number of terms we'll search for, is 2. These terms are $T_0 = 0$ and $T_1 = 3$.
- We won't specify most of H; only that $\ell = 4$, $H(T_0) = 0110 = 6$ and $H(T_3) = 0010 = 2$.
- Our return data are $\delta = 4$ bits long; let $b = 2$. We limit ourselves to $r = 4$.
- Let's consult an RNG to choose values of ζ for use in Paillier.

Let's Consult an RNG



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Source: `http://imgs.xkcd.com/comics/random_number.png`

Actual Example Setup

Great, we will randomly set $\zeta = 4$ for all encryptions.

Example Query

- Since $H(T_0) = 6$,

$$\begin{aligned}\mathcal{E}_6 &= \mathcal{E}(2^{0\cdot 2}) \\ &= 639\end{aligned}$$

- Similarly, since $H(T_1) = 2$,

$$\begin{aligned}\mathcal{E}_2 &= \mathcal{E}(2^{1\cdot 2}) \\ &= 359.\end{aligned}$$

- All other terms are encryptions of 0; we will write these as 1 even though they would in fact be distributed across a wide array of values in $(\mathbf{Z}/N^2\mathbf{Z})^\times$.

Example Response

Suppose, as in our example above, that the responder inputs, in order, are (T_0, D^0), (T_1, D^1), $(5, D^2)$, and (T_0, D^3), after which point the responder returns (perhaps another T_0 comes in, thus causing c_{T_0} to be greater than r). Here,

$$\begin{aligned}
D^0 &= 0000 = (D_0^0, D_1^0) = (00, 00),\\
D^1 &= 0110 = (D_0^1, D_1^1) = (01, 10),\\
D^2 &= 0111 = (D_0^2, D_1^2) = (01, 11),\\
D^3 &= 0010 = (D_0^3, D_1^3) = (00, 10),
\end{aligned}$$

Note that since 5 is not a search term, it will result in raising an encrypted zero to $D^2 = 7$; again, we're just going to write 1, even though the acutal algorithm may (will) have any encryption of 0 instead.

Example Response Matrix

The responder forms the matrix

Index	$\mathcal{Y}_0$	$\mathcal{Y}_1$	$\mathcal{Y}_2$	$\mathcal{Y}_3$
$\vdots$				
2	$\mathcal{E}_{\mathbf{2}}^{D_{\mathbf{0}}^{\mathbf{1}}} \mod N^{\mathbf{2}} = 359$	$\mathcal{E}_{\mathbf{2}}^{D_{\mathbf{1}}^{\mathbf{1}}} \mod N^{\mathbf{2}} = 256$	1	1
$\vdots$				
6	$\mathcal{E}_{\mathbf{6}}^{D_{\mathbf{0}}^{\mathbf{0}}} \mod N^{\mathbf{2}} = 1$	$\mathcal{E}_{\mathbf{6}}^{D_{\mathbf{1}}^{\mathbf{0}}} \mod N^{\mathbf{2}} = 1$	$\mathcal{E}_{\mathbf{6}}^{D_{\mathbf{0}}^{\mathbf{3}}} \mod N^{\mathbf{2}} = 1$	$\mathcal{E}_{\mathbf{6}}^{D_{\mathbf{1}}^{\mathbf{3}}} \mod N^{\mathbf{2}} = 396$
7	1	1	1	1
$\vdots$				

Example Responses

The only interesting responses are $\mathcal{Y}_0 = 359$, $\mathcal{Y}_1 = 256$, and $\mathcal{Y}_3 = 396$ (products are taken down columns).

Example Result

We decrypt to $Y_0 = 0100$, $Y_1 = 8 = 1000$ and $Y_3 = 2 = 0010$, and then run through the processing algorithm:

- Data For T_0: $M = 0011$
  - $X_1 = 0$:
    - $D_0 = (Y_0\&0011)/2^0 = 00$
    - $D_1 = (Y_1\&0011)/2^0 = 00$
  - $X_2 = 2$:
    - $D_0 = (Y_2\&0011)/2^0 = 00$
    - $D_1 = (Y_3\&0011)/2^0 = 10$

Example Result

We decrypted to $Y_0 = 0100$, $Y_1 = 8 = 1000$ and $Y_3 = 2 = 0010$, and then run through the processing algorithm:

- Data For T_1: $M = 1100$.
  - $X_1 = 6$:
    - $D_0 = (Y_0 \& 1100)/2^2 = 01$
    - $D_1 = (Y_1 \& 1100)/2^2 = 10$
  - $X_2 = 0$:
    - $D_0 = (Y_2 \& 1100)/2^2 = 00$
    - $D_1 = (Y_3 \& 1100)/2^2 = 00$

These results are precisely the data the responder had.*

*****: We cannot distinguish the fact that X_2 is a non-response from the possibility that X_2 represents an actual return of a datum $D = 0$ from the responder. In practice, one must avoid using $D = 0$ to eliminate this ambiguity