HW2

Adam Bowers, Dalton Cole, Sammie Bush

December 4, 2017

PseudoCode

Let l be the bit length of u&v, and let 1 be most significant bit and l least significant bit. Part 1 Party one randomly chooses functionality F=u¿v or u¡v He first computes

- for i=1 to l
  - $E_{pk}(u_i * v_i) \leftarrow SM(E_{pk}(u_i), E_{pk}(v_i))$
    first party one computes the product of current bit of v and u
  - if selected F: was u>v
    $W_i \leftarrow E_{pk}(u_i) * E_{pk}(u_i * v_i)^{n-1}$
    compute $u_i - u_i * v_i$ if there is any $W_i = 1$ we know for i $u_i = 1$ & $v_i = 0$
    $\Gamma_i \leftarrow E_{pk}(v_i - u_i) * E_{pk}(r_i)$ where $r_i \in Z_n$
    compute $v_i - u_i + r_i$
  - else
    $W_i \leftarrow E_{pk}(v_i) * E_{pk}(u_i * v_i)^{N-1}$
    compute $v_i - u_i * v_i$ if there is any $W_i = 1$ we know for i $v_i = 1$ & $u_i = 0$
    $\Gamma_i \leftarrow E_{pk}(u_i - v_i) * E_{pk}(r_i)$ where $r_i \in Z_n$
    compute $u_i - v_i + r_i$
  - $G_i \leftarrow E_{pk}(u_i \oplus v_i)$
    now compute xor of two bits if 0 they're the same so first 1 tells us first different bit
  - $H_i \leftarrow H_{i-1}^{s_i} * G_i$; where $s_i \in Z_n$ and $H_0 = E_{pk}(0)$
    now mask the xor based on previous bits and a random number, Hs will be 0 till first 1 then every term past that will be a random value based on s values.
  - $\Phi_i \leftarrow E_{pk}(-1) * H_i$
    shift the domain of by n-1 **Not sure what this is for? to mask output for party two? two make sure 1s are 0 and every other value is a random value?**
  - $L_i \leftarrow W_i * \Phi_i^{t_i}$ where $t_i \in Z_n$
    if W_i is 1 $\Phi_i^{t_i}$ will be 0 thus L_i will be 1 otherwise it will be some random value
- $\Gamma' \leftarrow \pi_1(\Gamma)$
- $L' \leftarrow \pi_2(L)$
  we permute the outputs so P2 can't tell anything from indices of bits
- send Γ'&L' to P2

Part 2
Party two computes

- $M \leftarrow D_{sk}(L'_i)$
  decrypt our Ls

- if there exists an $M_i = 1$

  $\alpha \leftarrow 1$

- else

  $\alpha \leftarrow 0$

  assign alpha based on L values. If there is a 1 we know the selected F function of P1 is true but not what it is

- $M'_i \leftarrow \Gamma_i^{\alpha'} for 1 \leq i \leq l$
  if we have a 1 in M then P1 gets *Gamma* back, otherwise they get random values

- send M' and $E_{pk}(\alpha)$ to P1

Part 3

- $M \leftarrow \pi^{-1}(M')$
  un permute M

- for i= 1 to l

  $\lambda_i \leftarrow M_i * E_{pk}(\alpha)^{N-r_i}$

  if alpha = 1 we subtract the random value added to L in part one
  to get either $v - u$ or $u - v$ depending on f chosen

  if F = $u > v$

  $E_{pk}(min(u,v)_i) \leftarrow E_{pk}(u_i) * \lambda_i$

  if $u > v$ $\lambda = v - u$ if $v > u$ otherwise 0
  therefore $u_i + \lambda_i$ gives us bit of highest value

  else

  $E_{pk}(min(u,v)_i) \leftarrow E_{pk}(v_i) * \lambda_i$

  if $v > u$ $\lambda = u - v$ if $u > v$ otherwise 0
  therefore $v_i + \lambda_i$ gives us bit of highest value

- concat $E_{pk}(min(u,v)_i)$ and Party one has $E_{pk}(min(u,v))$ as required

Two Phase Version

Let l be the bit length of u&v, and let 1 be most significant bit and l least significant bit. Furthermore let BD be a secure Binary Decomposition function & SM be a secure multiplication function

Phase one of the protocol returns E(0) if u is the minimum and E(1) if v is the minimum or equal to u.

(Minimum(E(u),E(v))

where E(u) and E(v) are encrypted bitwise Part 1

Party one randomly chooses functionality F=u¿v or u¡v He first computes

- for i=1 to l

- $E_{pk}(u_i * v_i) \leftarrow SM(E_{pk}(u_i), E_{pk}(v_i))$
  first party one computes the product of current bit of v and u
- if selected F: was u>v

  $$W_i \leftarrow E_{pk}(u_i) * E_{pk}(u_i * v_i)^{n-1}$$

  compute $u_i - u_i * v_i$ if there is any $W_i = 1$ we know for i $u_i = 1$ & $v_i = 0$
- else

  $$W_i \leftarrow E_{pk}(v_i) * E_{pk}(u_i * v_i)^{N-1}$$

  compute $v_i - u_i * v_i$ if there is any $W_i = 1$ we know for i $v_i = 1$ & $u_i = 0$
- $G_i \leftarrow E_{pk}(u_i \oplus v_i)$
  now compute xor of two bits if 0 they're the same so first 1 tells us first different bit
- $H_i \leftarrow H_{i-1}^{s_i} * G_i$; where $s_i \in Z_n$ and $H_0 = E_{pk}(0)$
  now mask the xor based on previous bits and a random number, Hs will be 0 till first 1 then every term past that will be a random value based on s values.
- $\Phi_i \leftarrow E_{pk}(-1) * H_i$
  shift the domain of by n-1 **Not sure what this is for? to mask output for party two? two make sure 1s are 0 and every other value is a random value?**
- $L_i \leftarrow W_i * \Phi_i^{t_i}$ where $t_i \in Z_n$
  if W_i is 1 $\Phi_i^{t_i}$ will be 0 thus L_i will be 1 otherwise it will be some random value

- $L' \leftarrow \pi_2(L)$
  we permute the output so P2 can't tell anything from indices of bits
- send L' to P2

Part 2
Party two computes

- $M \leftarrow D_{sk}(L'_i)$
  decrypt our Ls
- if there exists an $M_i = 1$

  $$\alpha \leftarrow 1$$

- else

  $$\alpha \leftarrow 0$$

  assign alpha based on L values. If there is a 1 we know the selected F function of P1 is true but not what it is
- send $E_{pk}(\alpha)$ to P1

Part 3

if alpha is one function is true 0 false, map that to
0 if u is min otherwise 1

- if F = $u > v$

  $$E_{pk}(result) \leftarrow E_{pk}(alpha)$$

- else

  $E_{pk}(result) \leftarrow (E_{pk}(alpha) * E_pk(-1))^{N-1}$
  if F was $u > v$ alpha already has 0 if u is min 1 if v
  for the other function we have to subtract 1 and to get 0 if u is minimum 1 otherwise
  return $E_pk(result)$

In phase two party one uses minimum function to get min index and returns that value
Return Minimum
P1 computes

- $E_{pk}(u) = BD(E_pk(u))$
- $E_{pk}(v) = BD(E_pk(v))$
- $MinIndice = Minimum(E_{pk}(u), E_{pk}(v))$
- for i=1 to l

  $\Gamma_i \leftarrow E_{pk}(v_i - u_i) * E_{pk}(r_i)$ where $r_i \in Z_n$
  compute $v_i - u_i + r_i$
- $\Gamma' \leftarrow \pi_1(\Gamma)$
- send Γ' and $MinIndice$ to P2

P2 computes

- $MinIndice \leftarrow D_{sk}(MinIndice)$
- $M'_i \leftarrow \Gamma_i^{(MinIndice}$ for $1 \le i \le l$
- return M'

P1 computes

- $M \leftarrow \pi^{-1}(M')$
- $\lambda_i \leftarrow M_i * E_{pk}(MinIndice)^{N-r_i}$ for i= 1 to l
  remove random terms from the difference so we can return the minimum
- $\Gamma \leftarrow E_{pk}(u) * \lambda$
  this returns u if u is minimum otherwise v
- $E_{pk}(min(E_{pk}(u), E_{pk}(v))) = \Gamma$

Thus party one has minimum encrypted element between u & v as required

Description

Pre-work Set a bit length for u and v.
Phase 1:
Goal: To compute whether u or v is the minimum

Part 1: First,Party 1 chooses a function. Either greater than or less than. Next, for each bit, we compute a list of masks L. Next, we permute these masks to create L'. Party 1 sends L' to

Party 2.

Part 2: Party 2 decrypts L' to create M. If any of the elements in M equals 1, α, else it equals 0. If 1 is present, we know the function chosen in the beginning is true. Party 2 then sends an encrypted copy of α

Part 3: If alpha is 1, function is true. If alpha is 0, it is false. Map that to 0 if u is min, otherwise, map to 1. If F was u ¿ v alpha already has 0 if u is min 1 if v for the other function we have to subtract 1 and to get 0 if u is minimum 1, otherwise, return Epk(result)

Phase 2:
Goal: To return the calculated minimum to the requesting party.

First, Party 1 computes Epk(u), Epk(v), and MinIndice. Next, for each bit, Party 1 computes Γ and Γ'. Lastly, Party 1 sends Γ' and MinIndice to Party 2.

Party 2 then decrypts the received MinIndice. With that and Γ, they compute M_i'. Party 2 then sends M' to Party 1.

Party 1 then computes M from M'. With that, Party 1 is able to compute λ and Γ. For each λ, Party 2 removes random terms from the differences so that the minimum can be found. If Γ is returned when computing it, u is the minimum. otherwise, v is the minimum.

Example

The above algorithm is implemented at: `https://github.com/drc14/Secure_Minimum`. The example was generated using the program implemented.

Minimum(E(u), E(v))
Party 1
Choose random functionality f: u > v
For 1 in 3:
$E(u_1) = E(1)$
$E(v_1) = E(1)$
$E(u_1 * v_1) = Secure_Multiplication(E(u_1), E(v_1)) = E(1)$
F: u > v:
$W_1 = E(0)$
$G_1 = E(u_1 \oplus v_1) = E(0)$
$H_1 = E(0)$
$\Phi_1 = E(12)$
$L_1 = W_1 * \Phi^(t_1) = E(0) * E(12)^8 = E(5)$
For 2 in 3:
$E(u_2) = E(1)$
$E(v_2) = E(0)$
$E(u_2 * v_2) = Secure_Multiplication(E(u_2), E(v_2)) = E(0)$
F: u > v:
$W_2 = E(1)$
$G_2 = E(u_2 \oplus v_2) = E(1)$
$H_2 = E(1)$

$\Phi_2 = E(0)$
$L_2 = W_2 * \Phi(t_2) = E(1) * E(0)^{10} = E(1)$
For 3 in 3:
$E(u_3) = E(1)$
$E(v_3) = E(1)$
$E(u_3 * v_3) = Secure_Multiplication(E(u_3), E(v_3)) = E(1)$
F: u > v:
$W_3 = E(0)$
$G_3 = E(u_3 \oplus v_3) = E(0)$
$H_3 = E(2)$
$\Phi_3 = E(1)$
$L_3 = W_3 * \Phi(t_3) = E(0) * E(1)^2 = E(2)$
$E(L) = E(5, 1, 2)$
$E(L') = E(2, 1, 5)$

Party 2
$M = D(E(L')) = [2, 1, 5]$
$\alpha = 1$ because 1 appears in M
$E(\alpha) = E(1)$

Party 1
$F : u > v$
Return $E(\alpha) = E(1)$

Party 1
E(u) = BD(E(7))
E(v) = BD(E(5))
Minimum_Index = $Mininimum(E(u), E(v)) = E(1)$
For $i = 1$ to 3: $Gamma_1 = E(v_1 - u_1) * E(r_1) = E(1 - 1) * E(5) = E(5)$
$Gamma_2 = E(v_2 - u_2) * E(r_2) = E(0 - 1) * E(10) = E(9)$
$Gamma_3 = E(v_3 - u_3) * E(r_3) = E(1 - 1) * E(8) = E(8)$
$\Gamma = E(5, 9, 8)$
$\Gamma' = \pi_1(\Gamma) = E(9, 8, 5)$

Party 2
$Min_Index = D(E(Min_Index)) = 1$
$M' = E(9, 8, 5)^1 = E(9, 8, 5)$

Party 1
$M = \pi^{-1}(M') = E(5, 9, 8)$
$\lambda = M * E(Min\_Index)^{N - r_i} = E(4, 11, 10)$
$\Gamma = E(u) * \lambda = E(5)$
$E(min(E(u), E(v))) = \Gamma = E(5)$