Oblivious Linear Evaluation (OLE) flavors from random OT

Here we sum up different OLE flavors, where some of them are needed for subprotocols of TLSNotary. All mentioned OLE protocol flavors are implementations with errors, i.e. in the presence of a malicious adversary, he can introduce additive errors to the result. This means correctness is not guaranteed, but privacy is.

Functionality $\mathcal{F}_{\text{ROT}}$

Note: In the literature there are different flavors of random OT, depending on if the receiver can choose his input or not. Here we that assume the receiver has a choice.

Define the functionality $\mathcal{F}_{\text{ROT}}$:

- The sender P_A receives the random tuple (t_0, t_1), where t_0, t_1 are κ-bit messages.
- The receiver P_B inputs a bit x and receives the corresponding t_x.

Random OLE

Functionality $\mathcal{F}_{\text{ROLE}}$

Define the functionality $\mathcal{F}_{\text{ROLE}}$ which maintains a counter k and which allows to call an Extend_k command multiple times.

- When calling Initialize set up the functionality for subsequent calls to Extend_k.
- When calling Extend_k: P_A receives (a_k, x_k) and P_B receives (b_k, y_k).

such that $y_k = a_k \cdot b_k + x_k$

Protocol Π_{ROLE}

1. Initialization:
   - P_B randomly samples $f \leftarrow \mathbb{F}$.
   - Both parties call $\mathcal{F}_{\text{ROT}}(f)$, so P_A knows t_0^i, t_1^i and P_B knows t_{f_i}.
   - With some PRF define: $s_{i,0}^k := \text{PRF}(t_0^i, k)$, $s_{i,1}^k := \text{PRF}(t_1^i, k)$
2. Extend_k: This can be batched or/and repeated several times.
   - P_A samples randomly $c_k \leftarrow \mathbb{F}$ and $e_k \leftarrow \mathbb{F}$
   - P_B samples randomly $d_k \leftarrow \mathbb{F}$.
   - P_A sends e_k and $u_i^k = s_{i,0}^k - s_{i,1}^k + c_k$ to P_B.
   - P_B defines $b_k = e_k + f$ and sends d_k to P_A.
   - P_A defines $a_k = c_k + d_k$ and outputs $x_k = \sum 2^i s_{i,0}^k - a_k \cdot e_k$
   - P_B computes

     $$\begin{aligned} y_i^k &= f_i\big(u_i^k + d_k\big) + s_{i,f_i}^k \\ &= f_i\big(s_{i,0}^k - s_{i,1}^k + c_k + d_k\big) + s_{i,f_i}^k \\ &= f_i \cdot a_k + s_{i,0}^k \end{aligned}$$

     and outputs $y_k = 2^i y_i^k$
3. Now it holds that $y_k = a_k \cdot b_k + x_k$.

Vector OLE

Functionality $\mathcal{F}_{\text{VOLE}}$

Define the functionality $\mathcal{F}_{\text{VOLE}}$ which maintains a counter k and which allows to call an Extend_k command multiple times.

- When calling Initialize, P_B inputs a field element b. This sets up the functionality for subsequent calls to Extend_k.
- When calling Extend_k: P_A receives (a_k, x_k) and P_B receives y_k.

such that $y_k = a_k \cdot b + x_k$

Protocol Π_{VOLE}

Note: This is the Π_{COPEe} construction from KOS16.

1. Initialization:
   - P_B chooses some field element b.
   - Both parties call $\mathcal{F}_{\text{ROT}}(b)$, so P_A knows t_0^i, t_1^i and P_B knows t_{b_i}.
   - With some PRF define: $s_{i,0}^k := \text{PRF}(t_0^i, k)$, $s_{i,1}^k := \text{PRF}(t_1^i, k)$
2. Extend_k: This can be batched or/and repeated several times.
   - P_A chooses some field element a_k and sends $u_i^k = s_{i,0}^k - s_{i,1}^k + a_k$ to P_B.
   - P_A outputs $x_k = \sum 2^i s_{i,0}^k$
   - P_B computes

$$\begin{aligned} y_i^k &= b_i \cdot u_i^k + s_{i,f_i}^k \\ &= b_i\big(s_{i,0}^k - s_{i,1}^k + a_k\big) + s_{i,f_i}^k \\ &= b_i \cdot a_k + s_{i,0}^k \end{aligned}$$

   and outputs $y_k = 2^i y_i^k$
3. Now it holds that $y_k = a_k \cdot b + x_k$.

Random Vector OLE

Functionality $\mathcal{F}_{\text{RVOLE}}$

Define the functionality $\mathcal{F}_{\text{RVOLE}}$ which maintains a counter k and which allows to call an Extend_k command multiple times.

- When calling Initialize, P_B receives a field element b. This sets up the functionality for subsequent calls to Extend_k.
- When calling Extend_k: P_A receives (a_k, x_k) and P_B receives y_k.

such that $y_k = a_k \cdot b + x_k$

Protocol Π_{RVOLE}

1. Initialization:
   - P_B chooses some field element f.
   - Both parties call $\mathcal{F}_{\text{ROT}}(f)$, so P_A knows t_0^i, t_1^i and P_B knows t_{f_i}.
   - P_A sends e to P_B and P_B defines $b = e + f$.

- With some PRF define: $s_{i,0}^k := \mathrm{PRF}(t_0^i, k)$, $s_{i,1}^k := \mathrm{PRF}(t_1^i, k)$

2. Extend_k: This can be batched or/and repeated several times.
   - P_A samples randomly $c_k \leftarrow \mathbb{F}$ and P_B samples randomly $d_k \leftarrow \mathbb{F}$.
   - P_A sends $u_i^k = s_{i,0}^k - s_{i,1}^k + c_k$ to P_B.
   - P_B sends d_k to P_A.
   - P_A defines $a_k = c_k + d_k$ and outputs $x_k = \sum 2^i s_{i,0}^k - a_k \cdot e$
   - P_B computes

$$\begin{aligned} y_i^k &= f_i\big(u_i^k + d_k\big) + s_{i,f_i}^k \\ &= f_i\big(s_{i,0}^k - s_{i,1}^k + c_k + d_k\big) + s_{i,f_i}^k \\ &= f_i \cdot a_k + s_{i,0}^k \end{aligned}$$

   and outputs $y_k = 2^i y_i^k$

3. Now it holds that $y_k = a_k \cdot b + x_k$.

OLE from random OLE

Functionality $\mathcal{F}_{\mathrm{OLE}}$
Define the functionality $\mathcal{F}_{\mathrm{OLE}}$. After getting input a from P_A and b from P_B return x to P_A and y to P_B such that $x + y = a \cdot b$.

Protocol Π_{OLE}
Both parties have access to a functionality $\mathcal{F}_{\mathrm{ROLE}}$, and call Extend_k, so P_A receives (a'_k, x'_k) and P_B receives (b'_k, y'_k). Then they perform the following derandomization:
- P_A sends $u_k = a_k + a'_k$ to P_B.
- P_B sends $v_k = b_k + b'_k$ to P_A.
- P_A outputs $x_k = x'_k + a'_k \cdot v_k$.
- P_B outputs $y_k = y'_k + b_k \cdot u_k$.

Now it holds that

$$\begin{aligned} y_k - x_k &= (y'_k + b_k \cdot u_k) - (x'_k + a'_k \cdot v_k) \\ &= (y'_k + b_k \cdot (a_k + a'_k)) - (x'_k + a'_k \cdot (b_k + b'_k)) \\ &= a_k \cdot b_k \end{aligned}$$