Distributed Multipoint Function

November 29, 2022

What do we have:

1. Big-state DMPF (errorless)
2. OKVS-based DMPF
   (a) OKVS through polynomial: errorless, inefficient
   (b) OKVS through [sparse matrix||dense matrix]: empirically small error, practically $\propto t$ Gen time and Eval/FullEval time independent to t.
   (c) Compared to batch-code based DMPF: approximately $\times 2$ faster FullEval
   (d) Some regular/nonregular optimization in PCG application.
   (e) Distributed key generation comparison

Definition 1. *The class of t-point functions, with input from $\{0,1\}^n$ and output from a group $\mathbb{G}$, is $\{f_{A,B}\}$ where A is a list of t distinct n-bit strings and B is a list of t $\mathbb{G}$ elements and* $f_{A,B}(x) = \begin{cases} 0_{\mathbb{G}} & \text{if } x \notin A \\ B[i] & \text{if } x = A[i], 1 \le i \le t \end{cases}$.

1 Big-state DMPF

1.1 The scheme

We display the big-state DMPF scheme in figure 1.1.

1.2 Distributed key generation

We display a distributed key generation protocol for the big-state DMPF in 1.2.

2 A new scheme of DMPF basing on OKVS

We provide a new strategy to distribute a multipoint function with a constraint on the input size: $n \le \lambda + 1$.

2.1 The raw scheme

The following algorithm in 2.1 is a distributed t-point function scheme with a control bit and without the convert layer. Each key k_b generated by $Gen(1^\lambda, A, B)$ can span a complete binary tree for party b, where each node contains a $(\lambda+1)$-bit string $s_b||t_b$ with s_b being a λ-bit seed and t_b being a control bit. The strings on the children of a node is obtained by first using s_b as PRG seed to get pseudorandom strings for both children, then applying a correction to both strings if the control bit t_b is 1. The correction is basing on the CW of the corresponding layer.

The correctness of the scheme is guaranteed by the invariance on the trees spanned by k_0 and k_1: if a node is not on any accepting path, then the strings on this node in two trees are identical. If a node is on an accepting path, then the seed strings are pseudorandom and independent, while the control bits must be distinct.

In the concrete pseudocode, A contains all accepting inputs, and $A^{(i)}$ contains all distinct length-i prefixes of such inputs. $S_b^{(i)}$ records all λ-bit seed strings at the nodes in the binary tree spanned by k_b, corresponding to the prefixes in $A^{(i)}$. $T_b^{(i)}$ records all control bits at those nodes.

procedure $\textsc{Gen}(1^\lambda, A)$
$t \leftarrow |A|, n \leftarrow |A[1]|$.
For $0 \le i \le n-1$, let $A^{(i)}$ be the sorted and deduplicated list of i-bit prefixes of strings in A. Specifically, $A^{(0)} = [\epsilon]$.
Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2t}$ be a public PRG.
Set $S_b^{(0)} = [r_b]$ and $T_b^{(0)} = [b||0^{m-1}]$ for $b = 0, 1$ where r_0, r_1 are sampled independently and randomly from $\{0,1\}^\lambda$.
for $i = 1$ to n **do**
Let $CW^{(i)}, S_0^{(i)}, T_0^{(i)}, S_1^{(i)}, T_1^{(i)}$ be empty lists.
for $l = 1$ to $A^{(i-1)}$ **do**
Parse $G(S_b^{(i-1)}[l]) = s_b^L||t_b^L||s_b^R||t_b^R$ for $b = 0, 1$ where $s_b^L, s_b^R \in \{0,1\}^\lambda$ and $t_b^L, t_b^R \in \{0,1\}^t$.
if $A^{(i-1)}[l]||0 \in A^{(i)}$ and $A^{(i-1)}[l]||1 \notin A^{(i)}$ **then**
$d \leftarrow$ the index of $A^{(i-1)}[l]||0$ in $A^{(i)}$.
Append $s_0^R \oplus s_1^R||t_0^L \oplus t_1^L \oplus e_d||t_0^R \oplus t_1^R$ to $CW^{(i)}$ where $e_d = 0^{d-1}10^{t-d}$.
else if $A^{(i-1)}[l]||1 \in A^{(i)}$ and $A^{(i-1)}[l]||0 \notin A^{(i)}$ **then**
$d \leftarrow$ the index of $A^{(i-1)}[l]||1$ in $A^{(i)}$.
Append $s_0^L \oplus s_1^L||t_0^L \oplus t_1^L||t_0^R \oplus t_1^R \oplus e_d$ to $CW^{(i)}$.
else $\triangleright$ both $A^{(i-1)}[l]||0$ and $A^{(i-1)}[l]||1$ are in $A^{(i)}$.
$d \leftarrow$ the index of $A^{(i-1)}[l]||0$ in $A^{(i)}$.
Randomly sample r from $\{0,1\}^\lambda$.
Append $r||t_0^L \oplus t_1^L \oplus e_d||t_0^R \oplus t_1^R \oplus e_{d+1}$ to $CW^{(i)}$.
end if
end for
Randomly and independently sample $t - |CW^{(i)}|$ strings from $\{0,1\}^{\lambda+2t}$.
If $i = n$ then skip the following for-loop.
for $l = 1$ to $|A^{(i-1)}|$ **do**
Parse $G(S_b^{(i-1)}[l]) = s_b^L||t_b^L||s_b^R||t_b^R$ for $b = 0, 1$.
Parse $T_b^{(i-1)}[l] \cdot CW^{(i)} = \Delta s_b||\Delta t_b^L||\Delta t_b^R$ for $b = 0, 1$.
if $A^{(i-1)}[l]||0 \in A^{(i)}$ and $A^{(i-1)}[l]||1 \notin A^{(i)}$ **then**
Append $s_b^L \oplus \Delta s_b$ to $S_b^{(i)}$ and $t_b^L \oplus \Delta t_b^L$ to $T_b^{(i)}$, for $b = 0, 1$.
else if $A^{(i-1)}[l]||1 \in A^{(i)}$ and $A^{(i-1)}[l]||0 \notin A^{(i)}$ **then**
Append $s_b^R \oplus \Delta s_b$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t_b^R$ to $T_b^{(i)}$, for $b = 0, 1$.
else
Append $s_b^L \oplus \Delta s_b$ to $S_b^{(i)}$ and $t_b^L \oplus \Delta t_b^L$ to $T_b^{(i)}$, for $b = 0, 1$.
Append $s_b^R \oplus \Delta s_b$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t_b^R$ to $T_b^{(i)}$, for $b = 0, 1$.
end if
end for
end for
for $l = 1$ to t **do** $\triangleright$ convert layer
Append $(-1)^{T_0^{(n)}[l][l]} \cdot \left(G_{convert}(S_0^{(n)}[l]) - G_{convert}(S_1^{(n)}[l]) - B[l]\right)$ to $CW^{(n+1)}$.
end for
Set $k_b \leftarrow (S_b^{(0)}, CW^{(1)}, CW^{(2)}, \cdots, CW^{(n)})$.
return (k_0, k_1).
end procedure
procedure $\textsc{Eval}(1^\lambda, b, k_b, x)$
Parse $k_b = ([s], CW^{(1)}, CW^{(2)}, \cdots, CW^{(n)})$.
$t \leftarrow$ number of rows of any $CW^{(i)}$.
$c \leftarrow b||0^{t-1}$.
for $i = 1$ to n **do**
Parse $c \cdot CW^{(i)} = \Delta s||\Delta t^0||\Delta t^1$ where $\Delta s \in \{0,1\}^\lambda$ and $\Delta t^0, \Delta t^1 \in \{0,1\}^t$.
Parse $G(s) = s^0||t^0||s^1||t^1$.
$s||c \leftarrow (s^{x[i]}||t^{x[i]}) \oplus (\Delta s||\Delta t^{x[i]})$
end for
return $s||\bigoplus_{j=1}^{t} c[j]$
end procedure

Figure 1: The big-state DMPF scheme

procedure DistributedGen$(1^\lambda, A_0, A_1, B_0, B_1)$ $\triangleright$ A_0 and A_1 are shares of A while B_0 and B_1 are shares of B.
  For $b = 0, 1$, party b sets $Seed_b^{(0)} = [r_b]$ and $Ind_b^{(0)} = [b||0^{t-1}]$ where $r_b \xleftarrow{\$} \{0,1\}^\lambda$.
  for $i = 1$ to n **do**
    //Local computation phase for $b = 0, 1$:
    Let $Sum_b^{(i)}$ be an empty list.
    for $l = 1$ to t **do**
      Append $\bigoplus_{1 \le k \le 2^{i-1}, Ind_b^{(i-1)}[k][l]=1} G(Seed_b^{(i-1)}[k])$ to $Sum_b^{(i)}$.
    end for
    //Online secure computation phase (two parties run a secure 2PC protocol for the following process):
    Let $CW^{(i)}$ be an empty list.
    for $l = 1$ to t **do**
      Let $\Delta s^L||\Delta t^L||\Delta s^R||\Delta t^R \leftarrow Sum_0^{(i)}[l] \oplus Sum_1^{(i)}[l]$ to $Sum^{(i)}$, where $\Delta s^L, \Delta s^R \in \{0,1\}^\lambda$ and $\Delta t^L, \Delta t^R \in \{0,1\}^t$.
    end for
    Reconstruct the list A. Let $A^{(j)}$ denote the sorted and deduplicated list of j-bit prefixes of strings in A.
    $d \leftarrow$ the index of $A^{(i-1)}[l]||0$ in $A^{(i)}$.
    if $|A^{(i-1)}| < l$ **then**
      Append a random $(\lambda + 2t)$-bit string to $CW^{(i)}$.
    else if $A^{(i-1)}[l]||0 \in A^{(i)}$ and $A^{(i-1)}[l]||1 \notin A^{(i)}$ **then**
      Append $\Delta s^R||\Delta t^L \oplus e_d||\Delta t^R$ to $CW^{(i)}$ where $e_d = 0^{d-1}10^{t-d}$.
    else if $A^{(i-1)}[l]||1 \in A^{(i)}$ and $A^{(i-1)}[l]||0 \notin A^{(i)}$ **then**
      Append $s_0^L \oplus s_1^L||t_0^L \oplus t_1^L||t_0^R \oplus t_1^R \oplus e_d$ to $CW^{(i)}$.
    else
      Randomly sample r from $\{0,1\}^\lambda$.
      Append $r||t_0^L \oplus t_1^L \oplus e_d||t_0^R \oplus t_1^R \oplus e_{d+1}$ to $CW^{(i)}$.
    end if
    //Local computation phase for $b = 0, 1$:
    Let $Seed_b^{(i)}, Ind_b^{(i)}$ be empty lists.
    for $k = 1$ to 2^{i-1} **do**
      Parse $Ind_b^{(i-1)}[k] \cdot CW^{(i)} = \Delta s_b||\Delta t_b^L||\Delta t_b^R$.
      $s^L||t^L||s^R||t^R \leftarrow G(Seed_b^{(i-1)}[k]) \oplus \Delta s_b||\Delta t_b^L||\Delta s_b||\Delta t_b^R$.
      Append s_L and then s_R to $Seed_b^{(i)}$.
      Append t^L and then t_R to $Ind_b^{(i)}$.
    end for
  end for
  //Local computation phase for $b = 0, 1$:
  Let $Sum_b^{(n+1)}$ be an empty list.
  for $l = 1$ to t **do**
    Append $\sum_{1 \le k \le 2^n, Ind_b^{(n)}[k][l]=1} G_{convert}(Seed_b^{(n)}[k])$ to $Sum_b^{(n+1)}$.
  end for
  //Online secure computation phase:
  Let $CW^{(n+1)}$ be an empty list.
  for $l = 1$ to t **do**
    Append $(-1)^{Ind_0^{(n)}[A[l]][l]} \cdot (Sum_0^{(n+1)}[l] - Sum_1^{(n+1)}[l] - B[l])$ to $CW^{(n+1)}$.
  end for
    $\triangleright$ Adding here one more local computation phase that corrects $G_{convert}(Seed_b^{(n)}[k])(1 \le k \le 2^n)$ basing on $Ind_b^{(n)}$ and $CW^{(n+1)}$ directly gives the result of $FullEval(1^\lambda, b, k_b) = \{Eval(1^\lambda, b, k_b, x)\}_{x \in \{0,1\}^n}$.
  Let $k_b \leftarrow (Seed_b^{(0)}, CW^{(1)}, \cdots, CW^{(n+1)})$.
  return k_b to party b.
end procedure

Figure 2: (Small-domain) Distributed key generation protocol for the big-state DMPF

procedure $\textsc{Gen}(1^\lambda, A, B)$

$t \leftarrow |A|$, $n \leftarrow |A[1]|$.

For $0 \le i \le n-1$, let $A^{(i)}$ be the sorted and deduplicated list of i-bit prefixes of strings in A. Specifically, $A^{(0)} = [\epsilon]$.

Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ be a public PRG.

Let $\mathbb{F} = \mathbb{F}_{2^{\lambda+2}}$ and let $map : \{0,1\}^{\lambda+2} \to \mathbb{F}$ be an efficiently computable and invertible 1-to-1 mapping.

Set $S_b^{(0)} \leftarrow [r_b]$ and $T_b^{(0)} = [b]$ for $b = 0,1$ where r_0, r_1 are sampled independently and randomly from $\{0,1\}^\lambda$.

for $i = 1$ to n **do**

Let $V, S_0^{(i)}, T_0^{(i)}, S_1^{(i)}, T_1^{(i)}$ empty lists.

for $l = 1$ to $|A^{(i-1)}|$ **do**

Parse $G(S_b^{(i-1)}[l]) = s_b^L || t_b^L || s_b^R || t_b^R$ for $b = 0,1$.

if $A^{(i-1)}[l]||0 \in A^{(i)}$ and $A^{(i-1)}[l]||1 \notin A^{(i)}$ **then**

$\Delta s || \Delta t^L || \Delta t^R \leftarrow s_0^R \oplus s_1^R || t_0^L \oplus t_1^L \oplus 1 || t_0^R \oplus t_1^R$.

Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^L \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0,1$.

else if $A^{(i-1)}[l]||0 \notin A^{(i)}$ and $A^{(i-1)}[l]||0 \in A^{(i)}$ **then**

$\Delta s || \Delta t^L || \Delta t^R \leftarrow s_0^L \oplus s_1^L || t_0^L \oplus t_1^L || t_0^R \oplus t_1^R \oplus 1$.

Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0,1$.

else

$\Delta s || \Delta t^L || \Delta t^R \leftarrow r || t_0^L \oplus t_1^L \oplus 1 || t_0^R \oplus t_1^R \oplus 1$ where r is randomly sampled from $\{0,1\}^\lambda$.

Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0,1$.

Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0,1$.

end if

Append $\Delta s || \Delta t^L || \Delta t^R$ to V.

end for

Let $CW^{(i)} \in \mathbb{F}^t$ be the coefficients of a $\mathbb{F}[X]$ polynomial $P_{CW^{(i)}}$ of degree less than t such that $P_{CW^{(i)}}(map(A^{(i-1)}[l])) = map(V[l])$ for all $1 \le l \le |A^{(i-1)}|$. (If $|A^{(i-1)}| < t$ then choose $P_{CW^{(i)}}$ to be a random polynomial that satisfies this condition.)

end for

▷ Add a convert layer.

Set $k_b \leftarrow [S_b^{(0)}, CW^{(1)}, CW^{(2)}, \cdots, CW^{(n)}]$.

return (k_0, k_1).

end procedure

procedure $\textsc{Eval}(1^\lambda, b, k_b, x)$

Parse $k_b = [[s], CW^{(1)}, CW^{(2)}, \cdots, CW^{(n)}]$.

Set $c \leftarrow b$.

for $i = 1$ to n **do**

Parse $G(s) = s^0 || t^0 || s^1 || t^1$.

Interpret $CW^{(i)}$ as a polynomial $P_{CW^{(i)}}$.

Parse $map^{-1}(P_{CW^{(i)}}(map(x[1...(i-1)]||0^{n-i+1}))) = \Delta s || \Delta t^0 || \Delta t^1$.

$s || c \leftarrow s^{x[i]} || t^{x[i]} \oplus (\Delta s || \Delta t^{x[i]}) \cdot c$.

end for

▷ Convert.

return s.

end procedure

Figure 3: The new DMPF scheme

Remark 2. *Actually P_{CW} doesn't need to be a polynomial. The property that $CW^{(i)}$ is an OKVS for pairs $\{(A^{(i-1)}[l], correction^l)\}_{1 \le l \le |A^{(i-1)}|}$ suffices, where $correction^l$ is $\Delta s || \Delta t^L || \Delta t^R$ computed at the node corresponding to $A^{(i-1)}[l]$.*

2.2 Efficiency analysis

Let N be the domain size of the class of t-point functions.

		$t\times$ DPF	MPFSS from (probabilistic) batch code[2][10][4][1]	Big-state DMPF	OKVS-DMPF
keysize		$t(\lambda+2)\log N$	$m\lambda\log(N/m)$	$t(\lambda+2t)\log N$	$\log N\times$OKVS code size
Gen()	Dominating operations	$2t\log N\times$ PRG	$2m\log(dN/m)\times$PRG Finding a matching of t inputs to m buckets	$2t\log N\times$PRG	$2t\log N\times$PRG, $\log N\times$OKVS Encoding
	Cheap operations	$O(t\lambda\log N)$	$O(m\lambda\log(dN/m))$	$O(t(\lambda+t)\log N)$	$O(t\lambda\log N)$
Eval()	Dominating operations	$t\log N\times$PRG	$d\log(dN/m)\times$PRG Finding all buckets an input is mapped to	$\log N\times$PRG	$\log N\times$PRG, $\log N\times$OKVS Decoding
	Cheap operations	$O(t\lambda\log N)$	$O(d\lambda\log(dN/m))$	$O((\lambda+t)\log N)$	$O(\lambda\log N)$
FullEval()	Dominating operations	$tN\times$PRG	$dN\times$PRG Finding the input sequence in every bucket	$N\times$PRG	$N\times$PRG, $N\times$ OKVS Decoding
	Cheap operations	$O(t\lambda N)$	$O(d\lambda N)$	$O((\lambda+t)N)$	$O(\lambda N)$

Table 1: Keysize and running time comparison for different DMPF constructions for domain size N, t accepting points and computational security parameter λ. We leave this table with the abstraction of (probabilistic) batch code in the second column and the abstraction of OKVS in the last column, and plug in concrete instantiations later. m in the second column stands for the number of buckets used in batch code, and d stands for the number of buckets that an input is mapped to (we only consider regular degree because this is the case in most instantiations).

OKVS construction	Error	Code size	Encoding time	Decoding time
Polynomial	no error	$t(\lambda+2)$	$O(t\log^2 t)$ (including $\mathbb{F}$-ops)	$O(t)$ (single) ; $O(\log^2 t)$ (batched) (including $\mathbb{F}$-ops)
3H-GCT[9] (oblivious; binary)	empiric	$(et+\hat{g}+\lambda_{stat})(\lambda+2)$	$O((\hat{g}+\lambda_{stat}+e)t)$ in total (no field ops are involved)	$(w+\hat{g}+\lambda_{stat})$ $\mathbb{F}$-+
3H-GCT[9] (oblivious; large field)	empiric	$(et+\hat{g})(\lambda+2)$	$O(et+\hat{g})$ for triangulation and $\hat{g}t$ $\mathbb{F}$-×	$\hat{g}$ $\mathbb{F}$-×
Ribbon[6]	empiric			
Ribbon[5] (binary)	(analytic) $2^{-\lambda_{stat}}$	$et(\lambda+2)$	$O(\lambda_{stat}^2)$ by SGAUSS in [5]	$\frac{e\lambda}{e-1}$ $\mathbb{F}$-+

Table 2: Different OKVS instantiations comparison for OKVS of t key-value pairs with key space $[N]$ and values from field $\mathbb{F}=\mathbb{F}_{2^{\lambda+2}}$. We consider the optimal-keysize construction which is a polynomial, and the presently fastest constructions in [9]. Dominated factors are neglected. λ_{stat} in the second row denotes the statistical parameter, and is implicit in the third row (the statistical error is small as long as the field is large enough). $w, \hat{g}$ and e are parameters given by [9]. Empirically $w=3, \hat{g}=2$ and $e=1.23$ works for $2^6\leq t\leq 2^{30}$. One comment is all of the OKVS's above are linear OKVS schemes, but the linearity is not necessary in our DMPF construction.

		$t\times$ DPF	MPFSS from batch code[2, 10, 4, 1]	Big-state DMPF	OKVS-DMPF
keysize		$t(\lambda+2)\log N$	$m(\lambda+2)\log(dN/m)$	$t(\lambda+2t)\log N$	$(et+\hat{g})(\lambda+2)\log N$
Gen()	Dominating operations	$2t\log N\times$ PRG	$2m\log(dN/m)\times$PRG d-way cuckoo hashing t keys to m buckets	$2t\log N\times$PRG	$2t\log N\times$PRG, $\hat{g}t\log N\times\mathbb{F}_{2^{\lambda+2}}$-×
	Cheap operations	$O(t\lambda\log N)$	$O(m\lambda\log(dN/m))$	$O(t(\lambda+t)\log N)$	$O(t\lambda\log N)$
Eval()	Dominating operations	$t\log N\times$PRG	$d\log(dN/m)\times$PRG	$\log N\times$PRG	$\log N\times$PRG, $\hat{g}\log N\times\mathbb{F}_{2^{\lambda+2}}$-×
	Cheap operations	$O(t\lambda\log N)$	$O(d\lambda\log(dN/m))$	$O((\lambda+t)\log N)$	$O(\lambda\log N)$
FullEval()	Dominating operations	$tN\times$PRG	$dN\times$PRG	$N\times$PRG	$N\times$PRG, $2\hat{g}N\times\mathbb{F}_{2^{\lambda+2}}$-×
	Cheap operations	$O(t\lambda N)$	$O(d\lambda N)$	$O((\lambda+t)N)$	$O(\lambda N)$

Table 3: Keysize and running time comparison for different DMPF constructions obtained by plugging in concrete instantiations of the abstract structures in table 1. Dominated factors are neglected. In this table, probabilistic batch code is achieved through cuckoo hashing[1, 10, 4], with two parameters d and m. Setting $m=1.5t$ and $d=3$ works for $t>200$ with failure probability $<2^{-40}$ as suggested in [1], and also for smaller t with failure probability approximately 2^{-20}. The OKVS used in OKVS-DMPF is from [9], the third row in table 2, with parameters $\hat{g}=2$ and $e=1.23$.

A common feature for the OKVS-DMPF and batch code DMPF is the evaluation (and full-domain evaluation) time does not increase with t. When t is not too big the evaluation time of OKVS-DMPF is much smaller than that of the batch code DMPF. For very small t the evaluation time of the big-state DMPF is comparable to the other two, but as t grows it becomes much larger than the other two.

The keysize of the OKVS-DMPF and batch code DMPF are comparable, and they are comparable to the keysize of the big-state DMPF when t is small. Again as t grows the keysize of the big-state DMPF becomes much larger than the other two, due to the t^2 term in its expression.

The *Gen*() time of all constructions grow with t. The *Gen*() time of OKVS-DMPF and batch code DMPF grows linearly with t, while that of big-state DMPF grows quadratically in with t.

Note that PBC and OKVS from [9] both have correctness errors, which lead to DMPF schemes with negligible failure probability in key generation. We may also use perfectly correct OKVS (for example, encoding to a polynomial) to obtain a DMPF with no failure in key generation, but its practical performance is much worse than the one with failure probability.

2.2.1 Concrete applications and parameters

We use $\mathrm{DMPF}_{t,N,\mathbb{G}}$ to denote a DMPF scheme for t-point functions with domain $[N]$ and output group $\mathbb{G}$.

Concrete application	Cost in terms of DMPF per correlation/execution	Typical DMPF parameters
PCG for OLE from Ring-LPN	seedsize $\propto$ DMPF.$keysize$ expand time $\propto$ DMPF.$FullEval()$	$t = 5^2, 16^2, 76^2$ $N = 2^{20}$
PSI-WCA	communication $\propto$ DMPF.$keysize$ client computation $\propto$ DMPF.$Gen()$ server computation $\propto$ DMPF.$Eval()$	$t =$any $N = 2^{128}$

Table 4: Concrete applications of DMPF.

PCG for OLE from Ring-LPN:

Background: hardness assumption $R^c\text{-}LPN_{R,1,\mathcal{HW}_t}$: Let $R = \mathbb{Z}_{|}[\mathbb{X}]/\mathbb{F}(\mathbb{X})$ for a prime p and $F(X) \in \mathbb{Z}_p$. $\mathcal{HW}_t$ denotes uniform distribution over t-sparse polynomials in R. An alternative of hardness of $R^c\text{-}LPN_{R,1,\mathcal{HW}_t}$ is $\{(\vec{a}, \langle 1||\vec{a}, \vec{e}\rangle)\} \approx_c \{(\vec{a}, r)\}_{r \leftarrow U(R)}$ where $\vec{a} = (a_1, \cdots a_{c-1}) \xleftarrow{\$} U(R^{c-1})$ and $\vec{e} = (e_0, e_1 \cdots, e_{c-1}) \leftarrow \mathcal{HW}_t^c$.

The PCG construction in [3] makes use of the fact that $\langle 1||\vec{a}, \vec{e_1}\rangle \cdot \langle 1||\vec{a}, \vec{e_2}\rangle = \langle (1||\vec{a}) \otimes (1||\vec{a}), \vec{e_1} \otimes \vec{e_2}\rangle$ while $\vec{e_1} \otimes \vec{e_2}$ consists of c^2 R-elements, with each entry's hamming weight at most t^2. One such PCG can be constructed by applying c^2 $DMPF_{t^2,2N,\mathbb{Z}_p}$'s.

If we change the hardness assumption to $R^c\text{-}LPN_{R,1,regular\text{-}\mathcal{HW}_t}$ with noise distribution $regular\text{-}\mathcal{HW}_t$ being the uniform distribution over all regular t-sparse polynomials in R, then each entry in $\vec{e_1} \otimes \vec{e_2}$ is a product of regular t-sparse polynomials, and can be shared through 2 sets of $\{DMPF_{k,2N/t,\mathbb{Z}_p}\}_{k=1,2,\cdots,t}$. DMPF constructed through DPF will benifit from the regularity of the noise distribution, while batch-code or OKVS-based DMPF being insensitive.

Noise distribution ($\vec{e}$)	Entropy	Total DMPF keysize	Total DMPF FullEval time
$(regular\text{-}\mathcal{HW}_t)^c$	$E_1 = c \cdot \log(\frac{N^t}{t^t})$	$c^2t^2\lambda\log(2N/t)$	$2c^2tN \times$PRG (DPF); $4c^2N \times$PRG+$4c^2\hat{g}N \times \mathbb{F}_{2^{2\lambda+2}}$-MUL (OKVS-DMPF)
$\mathcal{HW}_t^c$	$E_2 = c \cdot \log(\frac{N^t}{t!})$	$c^2t^2\lambda\log(2N)$	$2c^2t^2N \times$PRG (DPF); $2c^2N \times$PRG+$2c^2\hat{g}N \times \mathbb{F}_{2^{\lambda+2}}$-MUL (OKVS-DMPF)

Table 5: Comparison among different choices of noise distribution in module-LPN assumption, and their time and space costs using different DMPF constructions. Dominated factors are ignored. We only consider trivial DMPF construction by sum of DPFs, and our OKVS-based DMPF. In the regime $N \gg ct$, the batch-code-based DMPF has similar tendency as the OKVS-DMPF, and is hence ignored.

Yaxin **Does entropy gain leads to significant efficiency improvement?** Notice that under the same N, c and t, $E_2 > E_1$ and $E_2 - E_1 \approx ct\log e$. Now let's suppose c and N are always fixed and t_1, t_2 are the choices of t for the first and second distribution such that they reach the same entropy. When $N \gg t$ we have the relation $\frac{t_1}{t_2} \approx (1 + 1/\log N)$, which is not a big difference.

From table 5 we can see that the noise distribution $regular\text{-}\mathcal{HW}_t$ should be preferred if we instantiate DMPF in PCG for OLE through sum of DPF's, while $\mathcal{HW}$ should be preferred if we instantiate DMPF through the big-state, batch-code or OKVS-DMPF.

Now let's compare the seed size and expand time of PCG with different DMPF instantiations in fig. 4, where the naive (DPF) one has regular noise distribution. For extremely small t ($t < 8$), the big-state DMPF yields the best expand time, at the expense of slightly larger seed size. For $t \geq 8$ the seed size of the big-state DMPF becomes incomparable to others while the expand time of the big-state DMPF grows with t and exceeds that of the naive DPF construction when t is around 130, which is larger than the typical parameters.

The expand time of the batch-code or OKVS-DMPF doesn't grow with t, and the expand time of OKVS-DMPF is about $0.5\times$ that of the batch-code-DMPF. However the seed size yielded by OKVS-DMPF is usually larger than the batch-code-DMPF. When t is as small as 8, the seed size yielded by OKVS-DMPF is only slightly larger, but when t grows to the largest typical parameter 76, the OKVS-DMPF is about $\times 2$ of the seed size of the batch-code-DMPF.

In short, choosing the big-state DMPF for $t < 8$ and the OKVS-DMPF for $t \geq 8$ gives at least $\times 2$ acceleration on expand time over other choices with sacrifice on the keysize. There is a tradeoff between the batch-code and OKVS-DMPF in that the OKVS-DMPF always provides a $\sim \times 2$ acceleration on expand time, but a loss in seed size that when t is large it may blow up the seed size to $\sim \times 2$ that of the batch-code-DMPF.

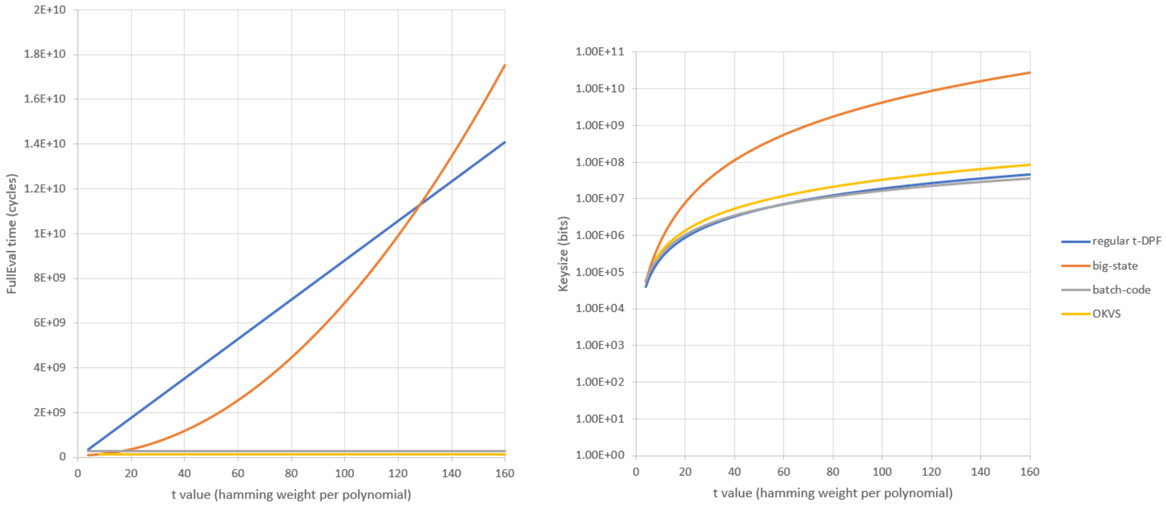

Figure 4: Full-domain Evaluation time and keysize of DMPF used in PCG for OLE[3] using four different DMPF constructions. Consider the security parameter $\lambda = 128$, the domain size $N = 2^{20}$ and various noise weights per R-element, from 4 to 160 (the typical weights per R-element in [3] are 5, 16 and 76). To obtain little failure probability, the OKVS-DMPF is only applicable for $t \geq 8$ as considered in [9]. PRG evaluation is modeled as two AES evaluations with AES evaluation time 1.3 cycles per byte. Field multiplications in OKVS-DMPF approach 0.3 cycles per byte [8] for the corresponding field. The actual expand time and seed size of PCG is $\sim \times c^2$ of that the FullEval time and key size of DMPF, where c is the compression factor.

Private Set Intersection - Weighted Cardinality:

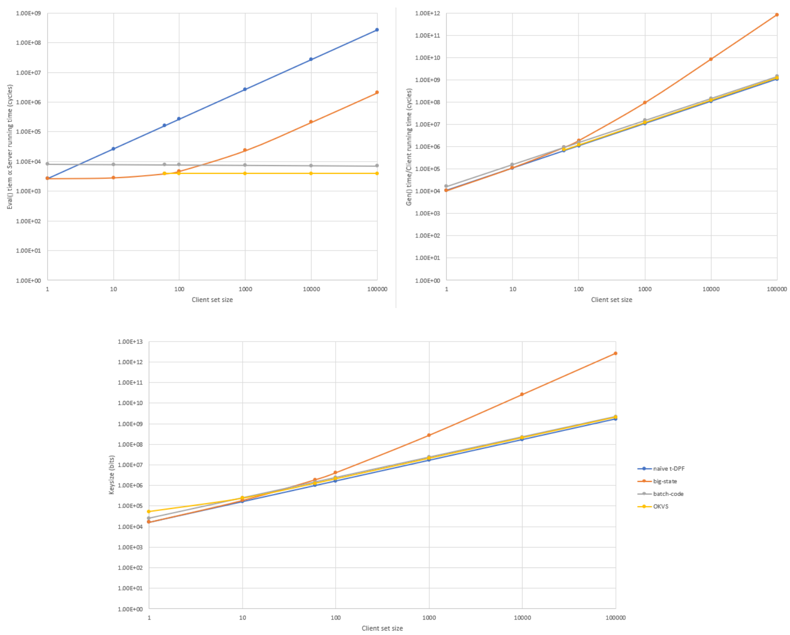

Figure 5: Key generation, evaluation time and keysize of DMPF used in PSI-WCA using four different DMPF constructions. Consider the security parameter $\lambda = 128$, the domain size $N = 2^{128}$ and various client set sizes from 1 to 100,000. To obtain little failure probability, the OKVS-DMPF is only applicable for $t \geq 64$ as considered in [9]. PRG evaluation is modeled as two AES evaluations with AES evaluation time 1.3 cycles per byte. Field multiplications in OKVS-DMPF approach 0.3 cycles per byte [8] for the corresponding field.

A short conclusion is using big-state DMPF for $t < 64$ and the OKVS-DMPF for $t \geq 64$ gives at $\sim \times 2$ faster

Eval() time and faster Gen() time compared to the naive and batch-code construction. The keysize ($\propto$ communication complexity) of our choice is usually smaller than the batch-code DMPF and slightly larger than the naive construction.

2.3 Security analysis

See appendix A.

2.4 Distributed key generation

Suppose party 0 and 1 each holds a share (A_0, B_0) and (A_1, B_1) for the secret t-point function $f_{A,B}$. In the sequel we display the distributed key generation for our DMPF construction that pushes PRG evaluations to local computation, with the cost that each party needs to locally compute $O(2^n)$ PRG evaluations (instead of tn online PRG evaluations).

Let $map : \{0,1\}^{2\lambda+2} \to \mathbb{F}_{2^{2\lambda+2}}$ be a public 1-to-1 mapping.
Let $H \in \mathbb{F}_{2^{2\lambda+2}}^{t\times 2^n}$ be a parity-check matrix for a $(2^n, 2^n - t, t+1)$ linear ECC with alphabet $\mathbb{F}_{2^{2\lambda+2}}$.
Party b samples $r_b \xleftarrow{\$} \{0,1\}^\lambda$ and sets $S_b^{(0)} = [r_b]$, $T_b^{(0)} = [b]$.
for $i = 1$ to n **do**
  // Local computation phase:
  Let $Word_b \leftarrow (0_{\mathbb{F}_{2^{2\lambda+2}}})^{2^n}$.
  for $x \in \{0,1\}^{i-1}$ **do**
    $Word_b[x] \leftarrow map(G(S_b^{(i-1)}[x]))$.
  end for
  ▷ According to the DMPF construction, $Word_0 - Word_1$ is of hamming weight $|A^{(i-1)}| \leq t$.
  Compute $Syndrome_b \leftarrow H \cdot Word_b$.
  // Secure 2PC phase:
  Let $Syndrome \leftarrow Syndrome_0 - Syndrome_1$.
  Reconstruct $A^{(i-1)}$ and recover $Word_0 - Word_1$ by syndrome decoding on $Syndrome$ with known error positions $A^{(i-1)}$.
  Knowing $Word_0 - Word_1$, compute $CW^{(i)}$ as in $Gen()$.
  // Local computation phase:
  for $x \in \{0,1\}^i$ **do** ▷ Local computation phase
    Apply correction basing on $S_b^{(i-1)}$, $T_b^{(i-1)}$ and $CW^{(i)}$ to obtain $S_b^{(i)}[x]$ and $T_b^{(i)}[x]$.
  end for
end for

Figure 6: Distributed key generation for the DMPF scheme when domain size N is feasible

The online phase computes the process in $Gen()$ excluding PRG evaluations, plus the process of decoding syndrome.

Instantiating the ECC with Reed-Solomon code in field $\mathbb{F}_{2^{2\lambda+2}}$, we have the first step of the 2PC phase equivalent to multiplying V^{-T} (V is the Vandermonde matrix for accepting points at the corresponding level) with a vector, while the second step of the 2PC phase equivalent to multiplying V^{-1} with a vector. Both can be achieved using $O(t \log^2 t)$ field-ops. This gives the total communication for distributing key generation $O(t \log^2 t \cdot \lambda \log N)$. Using the 2-level hashing idea gives some improvements in the first step, but asymptotically the same.

Another hope that reduces the asymptotic communication cost is to instantiate the ECC with OKVS row functions, namely, $H = [row(0)^T, \cdots r(1^n)^T]$ is of size $1.23t \times 2^n$. The linear system solved in the first step of 2PC phase is $H' \cdot \vec{x} = Syndrome$ where H' is the matrix obtained by restricting H to the accepting paths on the corresponding level. The linear system solved in the second step of 2PC phase is $H'^T \cdot CW = \Delta S || \Delta T$. The OKVS scheme [9] provides a fast algorithm to solve the second system that can also be used to solve the first one. Optimistically the total communication cost can be down to $O(t\lambda \log N)$, overlooking the cost for permuting matrix columns.

Unfortunately, batch-code based DMPF appears good in this aspect. The 2PC phase of distributed key generation contains first cuckoo hashing t elements to m buckets which requires $\tilde{O}(t)$ computation, and then compute the distributed key generation for DPFs in each bucket, which requires $O(m\lambda \log N)$ computation. If we plug in the practical parameter $m = O(t)$, then the total communication is $\tilde{O}(t) + O(t\lambda \log N)$.

2.5 All about the convert layer

Yaxin TBD: construct the convert layer for different output types:

1. $\mathbb{G} = (\{0,1\}^\lambda, \oplus)$
2. $\mathbb{G} = \{0,1\}$ (early termination)
3. $\mathbb{G} = \mathbb{F}$
4. General $\mathbb{G}$

and argue that they are secure.

2.6 Distributed Multi-inverval

Add another bit at each node and correct at each child.

3 RSA-PPRF and DtPF

3.1 RSA-based PPRF

We provide a construction for puncturable PRF basing on standard RSA assumption that has polynomial input and output domain size (M and S respectively), can puncture any subset and has near optimal punctured keysize.

- $pPRF.Gen(1^\kappa)$: Let $N = pq$ be a κ-bit RSA modulus. Let $M, S \in poly(\kappa)$ be the input and output domain sizes of the PRF respectively. Let $e_1, \cdots e_M$ be random κ-bit RSA exponents relatively prime to $\Phi(N)$.

  Output $msk = (N, \{e_i\}_{i\in[M]}, (p,q), u \xleftarrow{\$} [N], r \leftarrow R)$ where R is the distribution for the hardcore function below.
- $pPRF.Eval(msk, x)$: Output $hc(u^{1/e_x} \mod N; r)$ where r is an additional random input and hc is a hard-core function of $\log|S|$-bit span for the RSA function $\alpha \mapsto \alpha^{e_x}$. For example it can be the Goldreich-Levin hard-core function [7].
- $pPRF.Puncture(msk, T)$: Given $T \subseteq M$ and msk, output $k_T = (N, \{e_i\}_{i\in M}, T, u^{\Pi_{j\in T} 1/e_j} \mod N, r)$.
- $pPRF.PuncEval(k_T, x)$: Given $k_T = (N, \{e_i\}_{i\in M}, T, v, r)$ and $x \in T$, output $hc(v^{\Pi_{j\in T, j\neq x} e_j}; r)$. If $x \notin T$, output $\perp$.

Note that we may assume $\{e_i\}_{i\in[M]}$ and r are public random sequence so that it need not be included in msk or k_T. Then $|k_T| \approx \kappa + |T|\log M$ is near optimal. A downside for this RSA-based PPRF is one evaluation using the punctured key takes feasible but long time. Hence it is suitable for the applications where keysize is more crucial.

Efficiency analysis for $PuncEval$: An evaluation of $PuncEval(k_T, \cdot)$ takes roughly the same time as $|T| - 1$ RSA encryptions. In some occasions we evaluate $PuncEval(k_T, \cdot)$ on all points in T, which consumes $|T|\log|T|$ RSA encryptions using a simple recursive algorithm.

3.2 From RSA-PPRF to $1/poly$-secure DtPF

Theorem 3. *Given the RSA-based PPRF, we can construct:*

1. *$1/poly$-secure programmable distributed t-point function with 1-sided keysize $|k_T|$.*
2. *$1/poly$-secure distributed pseudorandom t-point function with keysize for two parties being $|msk|$, $|k_T|$.*
3. *Basing on 2, $1/poly$-secure DtPF with keysize for two parties being $|msk| + t\log S$, $|k_T| + t\log S$. (Yaxin TBD: how to permute?)*

4 From leaky DPF to secure DtPF

We can construct computationally secure PDtPF basing on $1/poly$-secure PDPF, in almost the same old way.

Theorem 4. *(Privacy amplification) Let $S = \binom{r+w}{w}$, L be as in the old construction and $q = r(t+\lambda-1)+1$. If there exists a poly-domain $O(1/qL)$-secure PDPF for point functions with output group $\mathbb{Z}_p$, domain size L and keysize K, then there is a (roughly) $e^{-\lambda/2}$-secure PtDPF with output group $\mathbb{Z}_p$, domain size S and keysize qK. (sketchy)*

Corollary 5. *If the PDPF is instantiated using RSA-PPRF then the keysize of the computationally secure PtDPF can be $\sim (\kappa + \log M) \cdot r(t+\lambda)$.*

References

[1] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. Pir with compressed queries and amortized query processing. Cryptology ePrint Archive, Paper 2017/1142, 2017. https://eprint.iacr.org/2017/1142.

[2] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector ole. Cryptology ePrint Archive, Paper 2019/273, 2019. https://eprint.iacr.org/2019/273.

[3] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. Cryptology ePrint Archive, Paper 2022/1035, 2022. https://eprint.iacr.org/2022/1035.

[4] Leo de Castro and Antigoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. Cryptology ePrint Archive, Paper 2021/580, 2021. https://eprint.iacr.org/2021/580.

[5] Martin Dietzfelbinger and Stefan Walzer. Efficient Gauss Elimination for Near-Quadratic Matrices with One Short Random Block per Row, with Applications, July 2019. arXiv:1907.04750 [cs].

[6] Peter C. Dillinger and Stefan Walzer. Ribbon filter: practically smaller than Bloom and Xor, March 2021. arXiv:2103.02515 [cs].

[7] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 25–32, New York, NY, USA, 1989. Association for Computing Machinery.

[8] Shay Gueron, Adam Langley, and Yehuda Lindell. Aes-gcm-siv: Specification and analysis. Cryptology ePrint Archive, Paper 2017/168, 2017. https://eprint.iacr.org/2017/168.

[9] Srinivasan Raghuraman and Peter Rindal. Blazing fast psi from improved okvs and subfield vole. Cryptology ePrint Archive, Paper 2022/320, 2022. https://eprint.iacr.org/2022/320.

[10] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-ole: Improved constructions and implementation. Cryptology ePrint Archive, Paper 2019/1084, 2019. https://eprint.iacr.org/2019/1084.

A Security proof for the DMPF scheme 2.1

A.0.1 Privacy

We argue the privacy of our scheme by a standard hybrid argument.

Algorithm 1 $Hyb_j(1^\lambda, b, A, B)$

Randomly sample $S_b^{(0)} = [s_b^{(0)}]$ from $\{0,1\}^\lambda$.
Set $T_0^{(0)} = [0]$ and $T_1^{(0)} = [1]$.
for $i = 1$ to n **do**
  if $i \leq j$ **then**
    Randomly sample $CW^{(i)} \leftarrow \mathbb{F}^t$.
    Let $S_b^{(i)}$ be an empty list.
    Interpret $CW^{(i)}$ as a polynomial $P_{CW^{(i)}}$.
    for $l = 1$ to $|A^{(i-1)}|$ **do**
      Parse $map^{-1}(P_{CW^{(i)}}(map(A^{(i-1)}[l]||0^{n-i+1}))) = \Delta s||\Delta t^L||\Delta t^R$.
      Parse $G(S_b^{(i-1)}[l]) = s_b^L||t_b^L||s_b^R||t_b^R$.
      if $A^{(i-1)}[l]||0 \in A^{(i)}$ and $A^{(i-1)}[l]||1 \notin A^{(i)}$ **then**
        Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^L \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$.
      else if $A^{(i-1)}[l]||0 \notin A^{(i)}$ and $A^{(i-1)}[l]||0 \in A^{(i)}$ **then**
        Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$.
      else
        Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$.
        Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$.
      end if
    end for
  else
    if $i = j + 1$ **then**
      Sample a list $S_{1-b}^{(i-1)}$ of $|A^{(i-1)}|$ independent and random λ-bit strings.
      Let $T_{1-b}^{(i-1)}$ be a list such that $T_{1-b}^{(i-1)}[l] = T_b^{(i-1)}[l] \oplus 1$.
    end if
    Let $V, S_0^{(i)}, T_0^{(i)}, S_1^{(i)}, T_1^{(i)}$ empty lists.
    for $l = 1$ to $|A^{(i-1)}|$ **do**
      Parse $G(S_b^{(i-1)}[l]) = s_b^L||t_b^L||s_b^R||t_b^R$ for $b = 0, 1$.
      if $A^{(i-1)}[l]||0 \in A^{(i)}$ and $A^{(i-1)}[l]||1 \notin A^{(i)}$ **then**
        $\Delta s||\Delta t^L||\Delta t^R \leftarrow s_0^R \oplus s_1^R||t_0^L \oplus t_1^L \oplus 1||t_0^R \oplus t_1^R$.
        Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^L \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0, 1$.
      else if $A^{(i-1)}[l]||0 \notin A^{(i)}$ and $A^{(i-1)}[l]||0 \in A^{(i)}$ **then**
        $\Delta s||\Delta t^L||\Delta t^R \leftarrow s_0^L \oplus s_1^L||t_0^L \oplus t_1^L||t_0^R \oplus t_1^R \oplus 1$.
        Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0, 1$.
      else
        $\Delta s||\Delta t^L||\Delta t^R \leftarrow r||t_0^L \oplus t_1^L \oplus 1||t_0^R \oplus t_1^R \oplus 1$ where r is randomly sampled from $\{0,1\}^\lambda$.
        Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0, 1$.
        Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0, 1$.
      end if
      Append $\Delta s||\Delta t^L||\Delta t^R$ to V.
    end for
    Let $CW^{(i)} \in \mathbb{F}^t$ be the coefficients of a $\mathbb{F}[X]$ polynomial P_{CW} such that $P_{CW}(map(A^{(i-1)}[l])) = map(V[l])$ for all $1 \leq l \leq |A^{(i-1)}|$. (If $|A^{(i-1)}| < t$ then choose P_{CW} to be a random polynomial that satisfies this condition.)
  end if
end for
▷ Add convert layer.
Output $[S_b^{(0)}, CW^{(1)}, CW^{(2)}, \cdots, CW^{(n)}]$.

Claim 6. *Suppose G is ϵ_G-secure against every n.u.p.p.t. adversary, then for every $\leq j \leq n$, every $b \in \{0,1\}$, every A containing t n-bit strings,B containing t $\mathbb{G}$ elements, and every n.u.p.p.t. adversary Adv,*

$$|\Pr[key \leftarrow Hyb_{j-1}(1^\lambda, b, A, B), Adv(1^\lambda, key) = 1] - \Pr[key \leftarrow Hyb_j(1^\lambda, b, A, B), Adv(1^\lambda, key) = 1]| \leq \epsilon_G |A^{(j-1)}|$$

Proof. Prove by contradiction. Assume Adv is a n.u.p.p.t adversary that for some $1 \leq j \leq n$, some $b \in \{0,1\}$, some

A and B,

$$|\Pr[key \leftarrow Hyb_{j-1}(1^\lambda, b, A, B), Adv(1^\lambda, key) = 1] - \Pr[key \leftarrow Hyb_j(1^\lambda, b, A, B), Adv(1^\lambda, key) = 1]| > \epsilon_G |A^{(j-1)}|$$

Then let's construct a n.u.p.p.t adversary Adv' which distinguishes $\{G(s)\}^{\otimes |A^{(j-1)}|}_{s \leftarrow U(\{0,1\}^\lambda)}$ from uniform distribution with advantage larger than $\epsilon_G |A^{(j-1)}|$, which implies some PRG-adversary distinguishing $\{G(s)\}_{s \in U(\{0,1\}^\lambda)}$ uniform distribution with advantage larger than ϵ_G.

Algorithm 2 PRG adversary $Adv'(1^\lambda, j, b, A, B, r)$ where $r \in \{0,1\}^{(2\lambda+2)|A^{(j-1)}|}$ is the challenge

---

Parse $r = r_0^1 || r_1^1 || r_0^2 || r_1^2 || \cdots || r_0^{|A^{(j-1)}|} || r_1^{|A^{(j-1)}|}$ where $|r_z^l| = \lambda + 1$.
Randomly sample $S_b^{(0)} = [s_b^{(0)}]$ from $\{0,1\}^\lambda$.
Set $T_0^{(0)} = [0]$ and $T_1^{(0)} = [1]$.
for $i = 1$ to n **do**
  if $i \le j-1$ **then**
    Randomly sample $CW^{(i)} \leftarrow \mathbb{F}^t$.
    Let $S_b^{(i)}$ be an empty list.
    Interpret $CW^{(i)}$ as a polynomial $P_{CW^{(i)}}$.
    for $l = 1$ to $|A^{(i-1)}|$ **do**
      Parse $map^{-1}(P_{CW^{(i)}}(map(A^{(i-1)}[l] || 0^{n-i+1}))) = \Delta s || \Delta t^L || \Delta t^R$.
      Parse $G(S_b^{(i-1)}[l]) = s_b^L || t_b^L || s_b^R || t_b^R$.
      if $A^{(i-1)}[l]||0 \in A^{(i)}$ and $A^{(i-1)}[l]||1 \notin A^{(i)}$ **then**
        Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^L \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$.
      else if $A^{(i-1)}[l]||0 \notin A^{(i)}$ and $A^{(i-1)}[l]||0 \in A^{(i)}$ **then**
        Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$.
      else
        Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$.
        Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$.
      end if
    end for
  else
    Define $R_0^l || R_1^l = \begin{cases} r_0^l || r_1^l & i = j \\ G(S_{1-b}^{(i-1)}[l]) & \text{else} \end{cases}$ for $1 \le l \le |A^{(i-1)}|$.
    Let $V, S_0^{(i)}, T_0^{(i)}, S_1^{(i)}, T_1^{(i)}$ empty lists.
    for $l = 1$ to $|A^{(i-1)}|$ **do**
      Parse $G(S_b^{(i-1)}[l]) = s_b^L || t_b^L || s_b^R || t_b^R$.
      Parse $R_0^l || R_1^l = s_{1-b}^L || t_{1-b}^L || s_{1-b}^R || t_{1-b}^R$.
      if $A^{(i-1)}[l]||0 \in A^{(i)}$ and $A^{(i-1)}[l]||1 \notin A^{(i)}$ **then**
        $\Delta s || \Delta t^L || \Delta t^R \leftarrow s_0^R \oplus s_1^R || t_0^L \oplus t_1^L \oplus 1 || t_0^R \oplus t_1^R$.
        Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^L \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0, 1$.
      else if $A^{(i-1)}[l]||0 \notin A^{(i)}$ and $A^{(i-1)}[l]||0 \in A^{(i)}$ **then**
        $\Delta s || \Delta t^L || \Delta t^R \leftarrow s_0^L \oplus s_1^L || t_0^L \oplus t_1^L || t_0^R \oplus t_1^R \oplus 1$.
        Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0, 1$.
      else
        $\Delta s || \Delta t^L || \Delta t^R \leftarrow r || t_0^L \oplus t_1^L \oplus 1 || t_0^R \oplus t_1^R \oplus 1$ where r is randomly sampled from $\{0,1\}^\lambda$.
        Append $s_b^L \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^L \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0, 1$.
        Append $s_b^R \oplus \Delta s \cdot T_b^{(i-1)}[l]$ to $S_b^{(i)}$ and $t_b^R \oplus \Delta t^R \cdot T_b^{(i-1)}[l]$ to $T_b^{(i)}[l]$ for $b = 0, 1$.
      end if
      Append $\Delta s || \Delta t^L || \Delta t^R$ to V.
    end for
    Let $CW^{(i)} \in \mathbb{F}^t$ be the coefficients of a $\mathbb{F}[X]$ polynomial P_{CW} such that $P_{CW}(map(A^{(i-1)}[l])) = map(V[l])$ for all $1 \le l \le |A^{(i-1)}|$. (If $|A^{(i-1)}| < t$ then choose P_{CW} to be a random polynomial that satisfies this condition.)
  end if
end for
▷ Add convert layer.
$key \leftarrow [S_b^{(0)}, CW^{(1)}, CW^{(2)}, \cdots, CW^{(n)}]$
Output $Adv(1^\lambda, key)$.

If r is a sample of $\{G(s)\}_{s\in U(\{0,1\}^\lambda}^{\otimes|A^{(j-1)}|}$, then $r_0^l||r_1^l = G(s^l)$ for some randomly sampled s^l, for all $1 \leq l \leq |A^{(j-1)}|$, which generates key in the same way as Hyb_{j-1}. Meanwhile if r is from the uniform distribution, the procedure generates key in the same way as Hyb_j.

Hence,

$$
\begin{aligned}
&|\Pr[key \leftarrow Hyb_{j-1}(1^\lambda, b, A, B), Adv(1^\lambda, key) = 1] - \Pr[key \leftarrow Hyb_j(1^\lambda, b, A, B), Adv(1^\lambda, key) = 1]| \\
=&|\Pr[s \leftarrow U(\{0,1\}^{\lambda|A^{(j-1)}|}), Adv'(1^\lambda, j, b, A, B, G^{\otimes|A^{(j-1)}|}(s) = 1] - \Pr[r \leftarrow U(\{0,1\}^{(2\lambda+2)|A^{(j-1)}|}), Adv'(1^\lambda, j, b, A, B, r) = 1]| \\
\leq&\epsilon_G|A^{(j-1)}|
\end{aligned}
$$

□

Together with the following two facts:

1. $\{key \leftarrow Hyb_0(1^\lambda, b, A, B)\} = \{k_b|(k_0, k_1) \leftarrow Gen(1^\lambda, A, B)\}$
2. $\{key \leftarrow Hyb_n(1^\lambda, b, A, B)\}$ is truly random.

we have the following security of the DMPF scheme:

Theorem 7. *Suppose $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ is ϵ_G-secure against every n.u.p.p.t. adversary. Then the scheme is a secure distributed t-point function scheme for the function family $f_{A,B} : \{0,1\}^n \to \mathbb{G}$ with key size $tn(\lambda + 1) + \lambda keysize_{convert}$ with secrecy $tn\epsilon_G + \epsilon_{convert}$.*

A.0.2 Correctness

The scheme has perfect correctness.