Adaptive-Input-Secure Updatable Private Set Union

Seongbong Choi[1], Jiseung Kim[2], and Hyung Tae Lee[1]

[1] Chung-Ang University, Republic of Korea
{welq2st, hyungtaelee}@cau.ac.kr
[2] Jeonbuk National University, Republic of Korea
jiseungkim@jbnu.ac.kr

Abstract. In multi-epoch deployments, private set union (PSU) operates in an adaptive-input loop: after observing the union at epoch t, the receiver may choose its next input for epoch $t + 1$. Liu et al. (EUROCRYPT 2026) formalized this multi-epoch adaptive-input setting for updatable private set intersection and provided an instantiation, but their framework does not extend to PSU. Meanwhile, existing PSU protocols are analyzed only in the single-shot setting.

We present the first semi-honest, adaptive-input-secure updatable PSU protocol supporting two-sided add/delete updates in the multi-epoch adaptive-input setting of Liu et al. Our construction is built around a new primitive, the *updatable oblivious key-value store* (uOKVS). Its defining rule, *distributional erasure*, requires each refresh to be distributed *identically* to a fresh static encoding of the current key set, rather than merely indistinguishably. We realize uOKVS by combining the Band-OKVS of Bienstock et al. with a PRF under a persistent key. The resulting refresh reuses a cached factorization, so its per-epoch encoding cost scales linearly rather than quadratically in the band width w. Building on this layer, we obtain a multi-epoch PSU protocol whose leakage is limited to set and update cardinalities, even against adaptive-input adversaries. We implement the protocol and benchmark it in a single-threaded setting. At $n = 2^{20}$ with per-epoch updates $\Delta = 55$ over 10 epochs, the online per-epoch wall-clock time is 1.41 s on LAN, yielding a 30.9×–98.9× speedup over prior static PSU protocols re-executed from scratch at each epoch.

Keywords: private set union · adaptive-input security · updatable OKVS · multi-epoch protocols.

1 Introduction

Private set union (PSU) allows a sender holding X and a receiver holding Y to compute $X \cup Y$, revealing only the union to the receiver and nothing to the sender. It underlies threat-intelligence sharing [19,32,30], abuse-blocklist aggregation [21], and cross-organization deduplication. In all of these settings, each input set is refreshed on a recurring schedule, such as hourly blocklist pulls and

continuous watchlist updates. A PSU deployment is therefore not a single execution, but a sequence of runs over many epochs, where at each epoch t the receiver observes $X_t \cup Y_t$.

1.1 Problem statement

Running a protocol over many epochs reveals information through changes from one epoch to the next. Such leakage arises in both private set intersection (PSI) and PSU and is inherent rather than avoidable, since it is already derivable from the outputs of the ideal functionality. An adversary that chooses each epoch's input adaptively, based on observations from earlier epochs, can exploit this inherent leakage across the execution. The security notion that captures such adaptively chosen inputs is *adaptive-input security*, the standard notion for reactive functionalities [9]. Liu et al. [24] first studied it for updatable PSI, yet the corresponding PSU setting remains unexplored.

The simplest way to reveal nothing beyond this inherent information is to run the protocol afresh at each epoch. Since each run is independent, no trace links one epoch to the next, and the resulting approach is secure. However, its cost scales with the total set size: even a small update forces a full recomputation over the entire set at every epoch. Indeed, the static PSU line [14,11,22] and its modern, performance-oriented successors [32,30,10,12,16,15,21,26,27] target only a single execution and do not address multi-epoch use.

To avoid full recomputation, a natural approach is to carry state across epochs so that the cost scales with the update size. This gain comes at a cost: once state is carried over, the data structure retains a cross-epoch trace that a fresh re-run would never leave. This trace is not granted by the ideal functionality, and an adversary can probe it through adaptively chosen inputs. The risk is sharper for PSU than for PSI. A PSI receiver learns $X_t \cap Y_t$, which remains within its own set Y_t, whereas a PSU receiver learns $X_t \cup Y_t$ and thus obtains the sender-only elements $X_t \setminus Y_t$ in the clear. The receiver can re-insert these cleartext elements in later epochs to probe the trace, giving it substantially more material to steer the execution than in PSI. For example, suppose the receiver learns an element z from an earlier union and later includes z in its own set. Then the next union contains z regardless of whether the sender keeps or deletes it, so the ideal output hides the sender's update. However, a stateful protocol may still expose this distinction through its carried-over state.

1.2 Our Solution

Our key insight is to carry state across epochs while forcing every re-encoding to be distributed identically to a fresh encoding of the current set. Whether an element was deleted or re-inserted, the same current state induces the same distribution, leaving no inter-epoch trace to probe. We call this property *distributional erasure* (DE) and formalize an updatable oblivious key-value store (uOKVS) satisfying it as a new primitive. The affine set encoding (ASE) of Liu et al. [24], used for PSI, encodes a set and supports only membership, whereas PSU must

Table 1: Updatable PSI/PSU landscape. Unknown: not proven and no known attack.

Work	PSI/PSU	Adaptive-Input	Update	Leakage
BMX22 [5]	PSI	Unknown	add-only	size
BMS$^+$24 [4]	PSI	No	add/del	size
ACF$^+$25 [3]	PSI	Unknown	add/del	size
ACG$^+$26 [1]	PSI	No	add/del	size
LTSQ26 [23]	PSI	Unknown	add/del	size
LMR$^+$26 [24]	PSI	Yes	add/del	size
Ours	**PSU**	**Yes**	**add/del**	**size**

reveal values in the clear. Supporting such cleartext reveals under adaptive input requires more than the update-simulatable refresh used for updatable PSI: any surviving inter-epoch trace could become a probeable signal. DE rules out this attack by requiring distributional identity with a fresh encoding, not merely indistinguishability of the update procedure. On top of a DE uOKVS, we build the first adaptive-input-secure updatable PSU, retaining the same scurity as fresh re-runs at a fraction of their per-epoch cost. Table 1 summarizes the comparison: our protocol is the only construction that delivers a union under adaptive input, with the same size-only leakage as prior updatable PSI.

1.3 Technical overview

We realize a uOKVS on top of Band-OKVS [7]. The state caches an LU factorization of the band matrix that covers every indexed row, including both active rows and dead rows left behind by past deletions. Each update touches this factorization only locally, reactivating a dead row, appending a new one, or clearing an active bit, so its cost grows with the size of the update rather than with the whole set.

Re-encoding reuses the cached factorization but recomputes the active values and draws fresh randomness for every free and dead-row slot before solving the system again. Because the band matrix stays full-rank, the result is distributed identically to a fresh encoding of the current active set, with no dependence on the path that produced it. The randomness of a deleted row is overwritten at the next re-encoding, so its trace disappears, which gives DE.

In the protocol, each party keeps its own uOKVS state, the sender over X_t and the receiver over Y_t. At each epoch, the change in the union decomposes into four update types: additions and deletions on either side. Each type needs a membership test to decide whether the element enters or leaves the union. Each test is settled by a single-item base-PSU query whose cost scales only with the update; since a cleartext OPRF would leak the matched position, the query reveals an element only via conditional opening.

1.4 Our Contributions

Our contributions are summarized as follows.

- We introduce DE, a property that forces each re-encoding to be distributed as a fresh one, leaving no trace between epochs. We formalize uOKVS by requiring this property and realize it as an updatable Band-OKVS (uBand-OKVS), whose refresh reuses a cached factorization and scales linearly in the band width w, in contrast to the quadratic cost of re-encoding from scratch. We also give a concrete deletion-trace attack against a naive state-reusing realization, showing that DE is necessary rather than merely convenient. These results are detailed in Section 3.
- Building on a DE uOKVS, we construct the first updatable PSU protocol secure against adaptively chosen inputs. It supports additions and deletions on both sides and leaks only set and update cardinalities. The protocol is presented in Section 4.
- We prove that the proposed protocol securely realizes the updatable PSU functionality against a semi-honest adversary, with leakage limited to set and update cardinalities. The security analysis appears in Section 5.
- We frame our construction so that any single-shot PSU can serve as a plug-in building block, formalized in Corollary 2. Section 6 presents one concrete instantiation and its evaluation, with additional measurements in Appendix A. At $n = 2^{20}$ with $\Delta = 55$, this instantiation completes each epoch 30.9×, 98.9×, and 49.1× faster than re-running JSZG24 [20], TBZ$^+$25 [30], and KLS26 [21] from scratch per-epoch.

2 Preliminaries

This section introduces the notation and recalls the building blocks used throughout the paper.

Notation. We write $[n] = \{1, \ldots, n\}$ for a positive integer n. For a finite map $I = \{(k_i, v_i)\}$, we write $\mathsf{supp}(I)$ for its key set $\{k_i\}$. $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{Z}$ denote a simulator, an adversary, and a multi-epoch environment, respectively. We use κ as the computational security parameter and λ as the statistical security parameter. $D_0 \overset{c}{\equiv} D_1$ indicates that the distributions D_0 and D_1 are computationally indistinguishable. We write $\otimes$ for the Kronecker product and I_ℓ for the $\ell \times \ell$ identity matrix. Throughout, $x \leftarrow S$ denotes uniform sampling from a finite set S, while $y \leftarrow \mathsf{Alg}(x)$ denotes the output of a randomized algorithm Alg on input x. We write $\mathbf{1}[\cdot] \in \{0, 1\}$ for the indicator function, which equals 1 when its argument holds and 0 otherwise.

We denote the sender's and receiver's sets at epoch t by X_t and Y_t, and their per-epoch update lists of $\mathsf{add}/\mathsf{del}$ operations by Δ_t^S and Δ_t^R. Each list decomposes into additions and deletions as $\Delta_t^S = \Delta_t^{S+} \sqcup \Delta_t^{S-}$ and $\Delta_t^R = \Delta_t^{R+} \sqcup \Delta_t^{R-}$. We write $\Delta_t = |\Delta_t^S| + |\Delta_t^R|$ for the total update size at epoch t, and $\Delta_{\max} = \max_t(|\Delta_t^{S+}| + |\Delta_t^{S-}| + |\Delta_t^{R-}|)$ for the maximum number of query-triggering update items over all epochs. Set elements are encoded as elements of $\mathbb{F}_2^\kappa$, and

elements longer than κ bits are first hashed by a collision-resistant function $H : \{0,1\}^* \to \mathbb{F}_2^\kappa$.

2.1 Building blocks

Oblivious key-value store (OKVS). An OKVS [25,17] over a key universe $\mathcal{K}$ and value space $V = \mathbb{F}_2^\kappa$ consists of two algorithms $(\mathsf{Encode}, \mathsf{Decode})$ with public parameters pp. The Encode algorithm takes pp and a finite map $I = \{(k_i, v_i)\}_{i=1}^N \subseteq \mathcal{K} \times V$, and outputs an encoding s. The Decode algorithm takes pp, an encoding s, and a key $k \in \mathcal{K}$, and outputs a value in V. It satisfies the following:

- *Correctness*: for any map $I' = \{(k_i, v_i)\}$ and every $(k_i, v_i) \in I'$,

$$\Pr[\mathsf{Decode}(\mathsf{pp}, \mathsf{Encode}(\mathsf{pp}, I'), k_i) = v_i] \geq 1 - \mathsf{negl}(\kappa).$$

- *Random decoding*: for any collection of non-stored keys $\{k_j^*\}$, the joint decoding distribution is computationally indistinguishable from independent uniform samples over V.

We may omit pp when they are clear from context.

A *linear OKVS* implements Encode by sampling uniformly from the solution set of a key-determined linear system $L \cdot s = \mathbf{v}$. We instantiate our construction with Band-OKVS [7], parameterized by band width w and expansion ϵ. The encoding length is $m = \lceil (1+\epsilon) N_{\max} \rceil$ for a deployment-time capacity $N_{\max}$.

Share-output oblivious pseudorandom function (so-OPRF). The so-OPRF functionality $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ secret-shares $F_k(x)$ between a key holder and an input provider as uniform XOR shares (z^0, z^1) satisfying $z^0 \oplus z^1 = F_k(x)$, so that neither party learns the PRF value. Figure 1 formalizes $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$.

Functionality $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$

Parameters: PRF $F : \mathcal{K} \times \{0,1\}^* \to V$, $V = \mathbb{F}_2^\kappa$. *Parties*: key holder H, input provider P.

- $\mathsf{Init}(k)$**.** From H: store $k \in \mathcal{K}$.
- $\mathsf{eval}(x)$**.** From P: sample $z^0 \leftarrow V$; set $z^1 := F_k(x) \oplus z^0$; deliver z^0 to H and z^1 to P.

Fig. 1: The so-OPRF functionality $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$.

Remark 1 (Why share-output, not cleartext, OPRF). The OKVS-OPRF construction of Kolesnikov et al. [22] and its descendants, such as RsOprf [29] in PSU mode, deliver $F_k(x)$ to the sender in the clear. On a value-carrying encoding whose decode yields F_k on stored keys, a corrupted sender could test

$F_k(x_j) \stackrel{?}{=} \mathsf{Decode}(L, x_j)$, recovering the membership index set $B_t = \{j : x_j \in Y_t\}$ and hence $X_t \cap Y_t$. This leakage goes beyond what the target updatable PSU functionality $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$ (Figure 4) permits. Prior approaches avoid this leakage via extra machinery at the cost of a round, such as an indication string [22], the enhanced functionality $\mathcal{F}_{\mathsf{ePSU}}$ [20], or Permute-and-Share [21]. In contrast, $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ eliminates this leakage by construction.

Secret-shared private equality test (ssPEQT). The ssPEQT functionality $\mathcal{F}_{\mathsf{ssPEQT}}$ takes inputs $m_0, m_1 \in \mathbb{F}_2^\kappa$ from the two parties and outputs additive shares of $\mathbf{1}[m_0 \oplus m_1 = 0^\kappa]$ to them. Let $\mathsf{Adv}^{\mathsf{ssPEQT}}$ denote the per-invocation simulation distance.

Oblivious transfer (OT). The $\binom{2}{1}$-OT ideal functionality $\mathcal{F}_{\mathsf{OT}}$ [28,13] takes sender inputs $(m_0, m_1) \in V^2$ and a receiver choice bit $c \in \{0, 1\}$, and delivers m_c to the receiver while revealing nothing about c to the sender. We use its standard preprocessing-model realization [6,18,8], and let $\mathsf{Adv}^{\mathsf{OT}}$ denote the corresponding simulation distance.

Remark 2 (Shared-choice OT from standard OT). Let $V_{\text{tag}} = \{0, 1\} \times V$ denote the tagged message space. The shared-choice OT functionality $\mathcal{F}_{\mathsf{OT}}^{\text{shared}}$ takes sender inputs $(M_0, M_1) \in V_{\text{tag}}^2$ and $b^S \in \{0, 1\}$, together with a receiver input $b^R \in \{0, 1\}$, and delivers $M_{b^S \oplus b^R}$ to the receiver and $\perp$ to the sender. $\mathcal{F}_{\mathsf{OT}}^{\text{shared}}$ is realized [15] in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model with simulation distance $\mathsf{Adv}^{\mathsf{OT}}$ by a swap-and-forward construction: the sender reorders its messages to $(M_{b^S}, M_{1 \oplus b^S})$ and invokes $\mathcal{F}_{\mathsf{OT}}$ on the receiver's choice b^R.

2.2 Security model

We follow the standard UC framework of Canetti [9] and consider *static semi-honest* corruption of at most one party, fixed at the beginning of the execution. The corrupted party follows the protocol, but its view is leaked to the adversary. For a protocol Π, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$, $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa)$ denotes the real-world transcript distribution. For a functionality $\mathcal{F}$ and a simulator $\mathcal{S}$, $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa)$ denotes the ideal-world transcript distribution. We say that Π realizes $\mathcal{F}$ if, for every pair of probabilistic polynomial-time (PPT) algorithms $(\mathcal{A}, \mathcal{Z})$, there exists a PPT simulator $\mathcal{S}$ such that $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa) \stackrel{c}{\equiv} \mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa)$.

We adopt *adaptive-input security* in the sense of Liu et al. [24]. At epoch $t = 0$, the environment $\mathcal{Z}$ fixes the initial sets (X_0, Y_0), and at each subsequent epoch $t \geq 1$, $\mathcal{Z}$ fixes the update lists (Δ_t^S, Δ_t^R) as a function of all messages observed in epochs $0, \ldots, t-1$. Inputs are fixed within an epoch but may change across epochs, unlike in the non-adaptive setting where all inputs are fixed up front. Appendix B explains why adaptive-input security is necessary for updatable PSU.

3 Updatable OKVS

This section introduces the uOKVS primitive, explains its design rationale, and gives a concrete construction.

3.1 uOKVS Ideal functionality $\mathcal{F}_{\mathsf{uOKVS}}$

The uOKVS ideal functionality $\mathcal{F}_{\mathsf{uOKVS}}$, formalized in Figure 2, maintains an active map I and an ever-stored key set J for the holder. At epoch t, the map I_t pairs each active key x with $F_k(t, x)$ under PRF key k, and J_t records every key seen so far. On every $\mathsf{ReEncode}$ call, the functionality outputs a fresh static encoding $\mathsf{Encode}(\mathsf{pp}, I_t)$. The $\mathsf{ReEncodeProtected}$ interface additionally accepts a one-epoch *protected set* P with $\mathsf{supp}(I_t) \subseteq P \subseteq J_t$ and outputs a fresh encoding over P.

Functionality $\mathcal{F}_{\mathsf{uOKVS}}$

Parameters: public parameter pp, capacity $N_{\max}$, encoding length $m = \lceil (1 + \epsilon)N_{\max} \rceil$, value space $V = \mathbb{F}_2^\kappa$, PRF F.
Party: holder H.
State: active map I, ever-stored set J, PRF key k, where $\mathsf{supp}(I) \subseteq J$ and $|J| \le N_{\max}$.

- $\mathsf{Init}(k, I_0)$**.** From H: store k, set $I := I_0$ and $J := \mathsf{supp}(I_0)$, and deliver $s_0 \leftarrow \mathsf{Encode}(\mathsf{pp}, I)$ to H.
- $\mathsf{Update}(\Delta)$**.** From H: apply each $(\mathsf{op}_j, x_j) \in \Delta$ by adding x_j to J and activating it in I for add, or by removing x_j from $\mathsf{supp}(I)$ for del while keeping x_j in J. Return $\perp$ on compatibility violation or if $|J| > N_{\max}$.
- $\mathsf{ReEncode}(t)$**.** From H: refresh $I := \{(x, F_k(t, x)) : x \in \mathsf{supp}(I)\}$ and deliver $s \leftarrow \mathsf{Encode}(\mathsf{pp}, I)$ to H.
- $\mathsf{ReEncodeProtected}(t, P)$**.** From H with a protected set P: deliver $s \leftarrow \mathsf{Encode}(\mathsf{pp}, \{(x, F_k(t, x)) : x \in P\})$ to H if $\mathsf{supp}(I) \subseteq P \subseteq J$, and $\perp$ otherwise.

Fig. 2: The uOKVS functionality $\mathcal{F}_{\mathsf{uOKVS}}$

3.2 Deletion-trace attack against non-DE realizations

We exhibit a concrete distinguisher against a stale-cell subclass of realizations that fail DE, even when combined with a so-OPRF. Appendix C gives a broader schedule-invariance statement that holds unconditionally.

The attack. A *stale-cell non-DE* realization Π_{uOKVS} repaints only active cells with the epoch-t PRF values at each refresh and leaves the cells of deleted or

never-stored keys at their stale values. The receiver starts with a set Y_0 of size n containing a target y^* known to the adversary. At epoch 1, the receiver deletes one key according to a hidden bit $b \in \{0,1\}$: it deletes y^* if $b = 0$ and a uniformly random $a \in Y_0 \setminus \{y^*\}$ if $b = 1$. A corrupted sender runs a distinguisher $\mathcal{D}_{\text{trace}}$ that observes only the two encodings L_0, L_1, with no PRF output revealed in the clear. It reads the target cell $j = h(y^*)$ and outputs $b' = 0$ if $L_0[j] = L_1[j]$ and $b' = 1$ otherwise. For clarity, we describe the single-cell case. Figure 3 contrasts the non-DE and DE encodings at the target cell.

L_0:	u_0	$F_k(0, y^*)$	u_2	$F_k(0, y_2)$	$F_k(0, y_3)$
L_1 (non-DE, b=0):	u_0	$F_k(0, y^*)$	u_2	$F_k(1, y_2)$	$F_k(1, y_3)$
L_1 (DE, b=0):	u'_0	u'_1	u'_2	$F_k(1, y_2)$	$F_k(1, y_3)$

Fig. 3: Cell trace at $j = h(y^*) = 1$ across two epochs, where y^* is deleted at epoch 1. Non-DE (red) retains the stale value $F_k(0, y^*)$, while DE (blue) resamples the cell to fresh randomness.

Theorem 1 (Deletion-trace distinguishing bound). *In the single-cell stale-cell model above, $\mathcal{D}_{\text{trace}}$ outputs the hidden bit b with probability at least $1-(n-1)/m-2^{-\kappa}$ against any stale-cell non-DE Π_{uOKVS}. Against any Π_{uOKVS} realizing $\mathcal{F}_{\mathsf{uOKVS}}$ where DE holds perfectly, the two cases induce identical views for $\mathcal{D}_{\text{trace}}$.*

Proof. In the non-DE case with $b = 0$, the target cell j is left unchanged, so $L_0[j] = L_1[j]$ unless another active key collides with cell j, which occurs with probability at most $(n-1)/m$ by a union bound over the remaining keys. With $b = 1$, the target remains active and cell j is repainted with the epoch-1 value, so $L_1[j] \neq L_0[j]$ except with probability $2^{-\kappa}$. Thus, $\mathcal{D}_{\text{trace}}$ outputs the correct bit except in one of these two events, giving success probability at least $1-(n-1)/m-2^{-\kappa}$. In the DE case, the refreshed encoding depends only on the current active set and fresh randomness. Hence, the views induced by $b = 0$ and $b = 1$ are identical at the target cell, so no distinguisher can beat guessing.

A cell-per-key simulation at $m = 563$, $|Y_0| = 200$, and $\kappa = 32$ over 4000 trials confirms this bound: the non-DE encoder lets $\mathcal{D}_{\text{trace}}$ recover the hidden bit with probability about 0.71, while the DE encoder stays near random guessing. The same intuition extends to Band-OKVS via window-level equality; we omit the corresponding bound since the single-cell attack already illustrates the need for DE.

3.3 Our uOKVS Construction over Band-OKVS

We instantiate uOKVS over Band-OKVS [7], which we call $\mathsf{uBand\text{-}OKVS}$. The concrete state at epoch t is $\phi_t = (\tilde{L}, \mathbf{v}_t, n_t, \mathsf{Act}_t, \mathsf{idx}_t)$, refining the ideal state

(I_t, J_t, k) of $\mathcal{F}_{\mathsf{uOKVS}}$. Here, $\tilde{L}$ is a cached LU factorization of the $\mathbb{F}_2$-coefficient band matrix H_t covering all ever-indexed rows, $n_t = |J_t|$ is the number of such rows, $\mathbf{v}_t$ stores their right-hand-side value slots, $\mathsf{Act}_t = \mathsf{idx}_t(\mathsf{supp}(I_t))$ is the active-row set, and $\mathsf{idx}_t : J_t \to [n_t]$ maps each ever-stored key to its row. The factorization is updated incrementally with each Update call.

Update processes an ordered update list $\Delta_t = [(\mathsf{op}_i, x_i)]$ sequentially, with no value field since values are generated from the PRF at the next ReEncode. For each item, it follows the add/del semantics of $\mathcal{F}_{\mathsf{uOKVS}}$.

- An *add-on-dead* operation, where $x_j \in J_t$ but the row $\mathsf{idx}_t(x_j)$ is inactive, marks the existing row $\mathsf{idx}_t(x_j)$ as active; the row receives $F_k(t, x_j)$ at the next ReEncode.
- An *add-on-new* operation, where $x_j \notin J_t$, returns $\perp$ if $n_t + 1 > N_{\max}$, and otherwise appends a fresh row to H_t, extends the cached factorization by incremental Gaussian elimination in $O(w^2)$ time, adds x_j to J_t, and indexes it as a new active row.
- A *del-active* operation, where the row $\mathsf{idx}_t(x_j)$ is active, marks this row as inactive, while keeping x_j in J_t and leaving its value slot to be randomized by the next ReEncode.
- An incompatible operation, such as adding an already active key or deleting a non-active key, returns $\perp$.

Thus, Update runs in amortized $O(|\Delta_t|w^2)$ time.

In the concrete construction, we write $\mathsf{ReEncode}(\phi, t)$ and $\mathsf{ReEncodeProtected}(\phi, t, P)$ to make the local state ϕ explicit; this state is implicit in the ideal functionality. $\mathsf{ReEncode}(\phi, t)$ solves the current system in four steps: it recomputes the active values $\mathbf{v}_{\mathrm{act}} \leftarrow \{F_k(t, k_i) : k_i \in \mathrm{active}(\phi)\}$, samples fresh free-variable values, samples fresh uniform values for each dead row, and back-substitutes through $\tilde{L}$ to produce the encoding s. This takes $O(mw\kappa)$ time, where $m = \lceil (1+\epsilon)N_{\max} \rceil$ is fixed at deployment time. The protected algorithm $\mathsf{ReEncodeProtected}(\phi, t, P)$ first checks $\mathsf{supp}(I_t) \subseteq P \subseteq J_t$, temporarily treats the rows indexed by $P \setminus \mathsf{supp}(I_t)$ as active during the solve, and reverts them before returning. If the check fails, it returns $\perp$.

DE relies on two algebraic facts: maintained full-rank factorization and dead-row marginalization. We first state the rank condition under which both facts are used.

Lemma 1 (Maintained factorization). *Let $\phi_t = (\tilde{L}, \mathbf{v}_t, n_t, \mathsf{Act}_t, \mathsf{idx}_t)$ be the protocol state at epoch t. For band width w and expansion ϵ in the Band-OKVS regime where the full-rank guarantee fails with probability at most $2^{-\lambda}$ over the placement seed ρ, every reachable state ϕ_t whose indexed keys are independent of ρ satisfies the following: $\tilde{L}$ is a valid factorization of the current $\mathbb{F}_2$-coefficient matrix $H_t \in \mathbb{F}_2^{n_t \times m}$ that encodes the band placement of all currently indexed rows, and $\mathrm{rank}_{\mathbb{F}_2}(H_t) = n_t$. The cell-wise lift acts on value vectors $s \in V^m = (\mathbb{F}_2^\kappa)^m$ via $H_t \otimes I_\kappa \in \mathbb{F}_2^{n_t\kappa \times m\kappa}$ and has full row rank $n_t\kappa$.*

Lemma 1 follows directly from the standard full-rank guarantee for band matrices from [7], where the rank-deficiency probability is exponentially small

in the band width. We instantiate at $w = 196$ and $\epsilon = 0.1$, where this probability is at most $2^{-\lambda}$ across our parameter range. We state the lemma separately to keep the analysis parametric in the underlying static OKVS.

The band-placement seed ρ is fixed once at Setup and remains unchanged throughout the lifetime of the state. Under adaptive-input security, the standard rank argument based on updates independent of ρ no longer applies. After seeing ρ as a public parameter, an adversary can rejection-sample keys to force rank collisions against the fixed seed, as discussed in Remark 3. On rank deficiency, the implementation aborts on the failed rank check. Each party's uOKVS state indexes only its local keys, so a failed rank check is a predicate only of those keys and the public seed ρ, not of the counterparty's set. Such executions are outside the rank-compatible condition used in the conditional security theorem and are treated as availability events rather than privacy failures. The security theorems in Section 5 hold conditional on rank-compatible executions.

Definition 1 (Rank-compatible execution). *An execution is* rank-compatible *if every internal rank check across all* $\mathsf{BaseEncode}$*,* Update*,* $\mathsf{ReEncode}$*, and* $\mathsf{ReEncodeProtected}$ *calls made by the* uOKVS *states of both parties over the* T *epochs succeeds.*

For value space $V = \mathbb{F}_2^\kappa$, the Band-OKVS coefficient matrix $H \in \mathbb{F}_2^{r\times m}$ acts on $s \in V^m$ through the lifted map $H \otimes I_\kappa$, so each row imposes one κ-bit vector equation. The next lemma shows that dead rows can be kept in the cached factorization without leaving a distributional trace: once their right-hand sides are sampled uniformly, they marginalize out and the resulting encoding is distributed as if only the active rows have been imposed.

Lemma 2 (Dead-row marginalization). *Let* $H \in \mathbb{F}_2^{r\times m}$ *be the Band-OKVS coefficient matrix at epoch* t*, with full row rank under Lemma 1. Let* $\mathsf{idx}_t : J_t \to [r]$ *be the index map maintained in the* $\mathsf{uBand\text{-}OKVS}$ *state and define* $\mathsf{Act}_t = \mathsf{idx}_t(\mathsf{supp}(I_t))$ *and* $\mathsf{Dead}_t = \mathsf{idx}_t(J_t \backslash \mathsf{supp}(I_t))$*. Let* H_A *and* H_D *be the submatrices of* H *indexed by* Act_t *and* Dead_t*, respectively. For* $V = \mathbb{F}_2^\kappa$*, if* $u \leftarrow V^{|\mathsf{Dead}_t|}$ *is sampled uniformly and then* s *is sampled uniformly from* $\{s \in V^m : H_A s = \mathbf{v}_{\mathrm{act}}, H_D s = u\}$*, then the marginal distribution of* s *is uniform over* $\mathcal{S}_{\mathrm{act}} = \{s \in V^m : H_A s = \mathbf{v}_{\mathrm{act}}\}$*. Thus, the resulting encoding has exactly the same distribution as* $\mathsf{Encode}(\mathsf{pp}, I_t)$ *on the current active map.*

Proof. In the flattened $\mathbb{F}_2$ formulation, the full constraint matrix is $H \otimes I_\kappa \in \mathbb{F}_2^{r\kappa\times m\kappa}$ acting on $\mathrm{vec}(s) \in \mathbb{F}_2^{m\kappa}$. By Lemma 1, $\mathrm{rank}_{\mathbb{F}_2}(H) = r$, and the Kronecker rank identity gives $\mathrm{rank}_{\mathbb{F}_2}(H \otimes I_\kappa) = r\kappa$. Consider the dead-row map $s \mapsto H_D s$ restricted to the active affine space $\mathcal{S}_{\mathrm{act}}$. The joint full row rank of H implies that this restricted map is surjective onto $\mathbb{F}_2^{|\mathsf{Dead}_t|\kappa}$, and every fiber is an equal-size affine coset of dimension $(m - |\mathsf{Act}_t| - |\mathsf{Dead}_t|)\kappa$ by rank–nullity. Sampling u uniformly over the codomain and then sampling s uniformly from the fiber above u yields s uniform over $\mathcal{S}_{\mathrm{act}}$. This is exactly the distribution of a fresh $\mathsf{Encode}(\mathsf{pp}, I_t)$ on the current active map.

The preceding lemmas give the distributional guarantee used below. The following adaptive hiding game models an honest holder's uOKVS state: the adversary may see prior encodings and auxiliary shares generated under the same key, but not the PRF key itself.

Definition 2 (Stateful adaptive $\mathcal{F}_{\mathsf{uOKVS}}$ hiding game). *A key $k \leftarrow \mathcal{K}_{\mathsf{PRF}}$ is sampled once and retained across all rounds. A stateful PPT adversary $\mathcal{A}$ receives (pp, ρ) but not k, and proceeds through adaptive rounds $t = 0, 1, \ldots, T$. At round t, $\mathcal{A}$ observes: (i) all previous challenge encodings $\{L_\tau\}_{\tau<t}$ and (ii) its entire auxiliary protocol view generated under the same k, including the corrupted party's $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ shares and prior $\mathcal{F}_{\mathsf{ssPEQT}}, \mathcal{F}_{\mathsf{OT}}^{\mathrm{shared}}$ views. $\mathcal{A}$ then outputs two equal-size ever-stored sets (A_t, D_t), or two protected sets, satisfying capacity and rank-compatibility conditions. A single hidden bit $b \leftarrow \{0,1\}$, fixed before round 0, determines*

$$L_t \leftarrow \begin{cases} \mathsf{Encode}(\mathsf{pp}, \{(x, F_k(\mathsf{dir}\|t\|x)) : x \in A_t\}) & \text{if } b = 0, \\ \mathsf{Encode}(\mathsf{pp}, \{(x, F_k(\mathsf{dir}\|t\|x)) : x \in D_t\}) & \text{if } b = 1, \end{cases}$$

where $\mathsf{dir} \in \{0,1\}$ is a direction bit, with 0 representing the receiver side and 1 representing the sender side. The next-round pair (A_{t+1}, D_{t+1}) may depend arbitrarily on $(L_0, \ldots, L_t)$ and on the auxiliary view. Define

$$\mathsf{Adv}_T^{\mathcal{F}_{\mathsf{uOKVS}}} = \max_{\mathcal{A}} \left| \Pr[b' = b] - \frac{1}{2} \right|,$$

where the maximum is over all PPT adversaries $\mathcal{A}$ in the above game.

The direction bit dir provides domain separation in the single-key abstraction of Definition 2. In the actual protocol, the sender and receiver use independently sampled PRF keys k^S and k^R, respectively; see Section 4.3. Thus, dir is only a conservative additional tag, since key independence already prevents cross-direction collisions. Once the direction is fixed, we write $F_k(t, x)$ for $F_k(\mathsf{dir}\|t\|x)$.

Lemma 3 (Adaptive hiding under rank compatibility). *For our uBand-OKVS construction, restricted to rank-compatible executions,*

$$\mathsf{Adv}_T^{\mathcal{F}_{\mathsf{uOKVS}}} \leq \mathsf{Adv}_F^{\mathsf{PRF}}(q) + q \cdot 2^{-\kappa},$$

where q is the total number of F_k queries in the game, including those used in challenge encoding queries and the adversary's auxiliary queries through $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$, and $\mathsf{Adv}_F^{\mathsf{PRF}}(q)$ is the adaptive multi-query PRF distinguishing advantage. The additive term bounds the accidental agreement between non-stored decodings and cached values.

Proof. PRF hop. We build an adaptive multi-query PRF distinguisher $\mathcal{B}$ from $\mathcal{A}$. $\mathcal{B}$ answers every F_k query through its oracle and maintains a lazy table with one consistent value for each input $\mathsf{dir}\|t\|x$. Thus, if a key appears both in a challenge encoding and in an auxiliary $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ query, it receives the same value, so the

relation $\mathsf{Decode}(L_t, x) = F_k(t, x)$ remains well defined. The epoch and direction tags separate inputs across epochs and directions. Replacing F_k with a random function changes the view by at most $\mathsf{Adv}_F^{\mathsf{PRF}}(q)$, where q counts all PRF inputs queried in the transcript, including challenge-encoding values, auxiliary $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ queries, and initialization values.

Marginal-uniformity hop. After the PRF hop, all right-hand sides used in challenge encodings are uniform, subject only to the consistency enforced by the lazy table. Fix a rank-compatible challenge set A_t and let H_{A_t} be its coefficient matrix. By Lemma 1, H_{A_t} has full row rank. For any fixed $s_0 \in V^m$, $\Pr[u_{A_t} = H_{A_t} s_0] = |V|^{-|A_t|}$ because u_{A_t} is uniform over $V^{|A_t|}$. Conditioned on this event, a uniform sample from $\{s : H_{A_t} s = u_{A_t}\}$ equals s_0 with probability $|V|^{-(m-|A_t|)}$. Therefore,

$$\Pr_{u_{A_t}, s}\left[s = s_0\right] = |V|^{-m},$$

so the encoding is uniform over V^m and independent of the key set A_t. The same argument applies to D_t. For protected challenges, Lemma 2 shows that the uniformly sampled dead-row values marginalize out, giving the same active-map distribution. Thus, in the random-function hybrid, the challenge encodings under $b = 0$ and $b = 1$ are identically distributed.

Auxiliary $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ consistency. The corrupted party may query $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ under the same persistent key on arbitrary inputs, including elements of the challenge sets and strings outside them. In the random function hybrid, each query returns uniform XOR shares of the cached random value, consistent with the cached encoding value on shared inputs. Since at most one party is corrupted, the adversary sees only one share and never learns the PRF value in the clear. Thus it cannot perform the cleartext check $F_k(t, x) \stackrel{?}{=} \mathsf{Decode}(L_t, x)$ that would distinguish stored challenge keys from non-stored ones. For inputs outside the encoded map, the decoding is uniform and independent of the hidden bit by the random-decoding property, and it matches the cached random value only with probability $2^{-\kappa}$. A union bound over the at most q such queries contributes an additive $q \cdot 2^{-\kappa}$ term. Hence, the auxiliary view adds no distinguishing advantage beyond the PRF hop and this collision term, which proves the lemma.

Lemma 3 reduces the stateful adaptive hiding game to adaptive multi-query PRF security and single-epoch Band-OKVS full-rank, with epoch-and-direction domain separation $\mathsf{dir}\|t\|x$ licensing the persistent key.

Theorem 2 (uBand-OKVS realizes $\mathcal{F}_{\mathsf{uOKVS}}$). *Let uBand-OKVS be the construction of Section 3.3, with placement seed ρ fixed at* Setup *and published. Assume that the Band-OKVS parameters are chosen so that each relevant rank-deficiency event occurs with probability at most $2^{-\lambda}$, and let Q_{rank} denote the total number of rank checks during the execution.*

(a) Conditional form. *For any $T = \mathsf{poly}(\lambda)$ calls to* ReEncode/ReEncodeProtected *issued by an adaptive-input environment, conditional on the execution being rank-compatible in the sense of Definition 1, the construction realizes $\mathcal{F}_{\mathsf{uOKVS}}$*

in the stateful adaptive hiding game of Definition 2 with

$$\mathsf{Adv}_{T}^{\mathcal{F}_{\mathsf{uOKVS}}} \leq \mathsf{Adv}_{F}^{\mathsf{PRF}}(q) + \mathsf{negl}(\lambda),$$

where the negligible term accounts for dummy-key collisions, the $q \cdot 2^{-\kappa}$ *collision term of Lemma 3, and* Encode/Decode *correctness slack.*

(b) Seed-independent unconditional form. *If the honest-party updates are drawn independently of the published seed* ρ*, then Lemma 1 bounds each rank-deficiency event by* $2^{-\lambda}$*. A union bound over the* Q_{rank} *rank checks gives total rank-failure probability at most* $Q_{\mathrm{rank}} \cdot 2^{-\lambda}$*. In this case, the construction realizes* $\mathcal{F}_{\mathsf{uOKVS}}$ *unconditionally with*

$$\mathsf{Adv}_{T}^{\mathcal{F}_{\mathsf{uOKVS}}} \leq \mathsf{Adv}_{F}^{\mathsf{PRF}}(q) + Q_{\mathrm{rank}} \cdot 2^{-\lambda} + \mathsf{negl}(\lambda).$$

Proof. DE hop. Consider a rank-compatible execution. On input (ϕ_t, t), ReEncode recomputes the active right-hand sides as $\{F_k(t, x)\}$, samples fresh uniform values for all dead rows and free variables, and back-substitutes through the cached LU factorization. By the uniform-solution property of Band-OKVS and Lemma 2, the resulting encoding is distributed exactly as a fresh $\mathsf{Encode}(\mathsf{pp}, I_t)$. For $\mathsf{ReEncodeProtected}(\phi_t, t, P)$, the rows indexed by $P \setminus \mathsf{supp}(I_t)$ are treated as active only for the duration of the call, while all remaining dead rows are marginalized in the same way. Thus, the concrete re-encoding step matches the corresponding ideal output distribution.

Hiding hop. Lemma 3 gives adaptive hiding under rank compatibility, with loss at most $\mathsf{Adv}_{F}^{\mathsf{PRF}}(q) + q \cdot 2^{-\kappa}$. After the PRF hop in that lemma, the uniform-solution argument makes each challenge encoding depend on the encoded set only through its size; Lemma 2 gives the corresponding marginal identity for protected encodings. Rank compatibility also ensures that concrete BaseEncode/Update calls and the ideal functionality follow the same $\perp$ branch. Together with the standard negligible terms from dummy-collisions and Encode/Decode correctness, this proves form (a).

Unconditional form. For honest updates independent of ρ, Lemma 1 bounds each relevant rank-deficiency event by $2^{-\lambda}$. A union bound over the Q_{rank} rank checks gives total rank-failure probability at most $Q_{\mathrm{rank}} 2^{-\lambda}$. Conditioning on the complement gives form (a), and adding the rank-failure probability yields the unconditional bound in form (b).

Remark 3 (Chosen-key DoS is an availability event). A holder that controls its input keys can force rank deficiency by rejection-sampling $w + 1$ keys whose Band-OKVS rows are supported within the same w columns. The resulting submatrix has $w + 1$ rows but at most w columns, and hence cannot have full row rank. Under a uniform-placement heuristic, finding such keys requires work exponential in w in the worst case, so we treat this as a qualitative availability risk rather than assigning it a calibrated concrete cost. Such an execution is not rank-compatible: the implementation aborts on the failed rank check, and our conditional security theorem excludes this branch. In the seed-independent unconditional form, the same event is charged to the rank-failure probability.

Systems that require liveness can re-sample the placement seed and rebuild the state after a rank failure; this engineering variant is orthogonal to the security claims analyzed here.

New-key capacity A uBand-OKVS instance is parameterized by an *ever-stored* bound $N_{\max}$, which counts both active and dead keys. The encoding length is $m = \lceil (1+\epsilon)N_{\max} \rceil$, the size for which Lemma 1 guarantees full row rank at $\epsilon = 0.1$ and $w = 196$. The initial ever-stored set consumes $|J_0| \le N_{\max}$ slots. Each subsequent add on an unseen key consumes one additional slot, whereas reactivations and deletions consume none. At the deployment high-water mark $|J_0| \approx 2^{20}$ and $N_{\max} \approx 1.15 \cdot 10^6$, the approximately 10^5 free slots support about 50 new keys per epoch for about 2000 epochs before re-initialization; the one-time re-initialization cost of about 2.7 s amortizes to about 1.4 ms per epoch.

The rank guarantee is stated on the full ever-indexed matrix, with at most $N_{\max}$ rows embedded in $m = \lceil (1+\epsilon)N_{\max} \rceil$ columns. The expansion $\epsilon N_{\max}$ is a *column* expansion: it provides extra encoding cells, beyond the row count, that supply the free-variable slack required by Lemma 1. It is not a row-capacity reserve.

4 Updatable PSU from uOKVS

Building on the $\mathcal{F}_{\mathsf{uOKVS}}$ primitive of Section 3, we construct an adaptive-input-secure updatable PSU protocol with two-sided add/delete updates.

4.1 Target functionality $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$

We define the target functionality $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$. The standard semi-honest PSU functionality $\mathcal{F}_{\mathsf{PSU}}$ takes inputs X from the sender and Y from the receiver, leaks $|X|, |Y|$, and outputs $X \cup Y$ to the receiver. Our updatable counterpart, shown in Figure 4, is its natural *two-sided* multi-epoch lift. Each party supplies an update list per epoch, and the receiver obtains the running union $X_t \cup Y_t$.

Per-epoch leakage is restricted to set sizes $(|X_t|, |Y_t|)$ and update sizes $(|\Delta_t^{S\pm}|, |\Delta_t^{R\pm}|)$. This profile is inherent to any non-padded multi-epoch protocol and extends the standard leakage convention from updatable PSI [24,4,1] to the PSU setting. Formally, the per-epoch leakage function is

$$\mathcal{L}(X_t, Y_t, \{\Delta_\tau\}_{\tau \le t}) = \big(|X_t|, |Y_t|, |\Delta_t^{S+}|, |\Delta_t^{S-}|, |\Delta_t^{R+}|, |\Delta_t^{R-}|\big).$$

Beyond $\mathcal{L}(\cdot)$, the sender learns nothing more, in particular no membership index set $B_t = \{j : x_j \in Y_t\}$, no element of $X_t \cap Y_t$, and no element of $\Delta_t^{R-} \cap X_t$. Hiding these cardinalities would require padding every batch to a public upper bound, at the corresponding bandwidth cost.

Remark 4 (Deletion is not knowledge revocation). Since PSU reveals each $x \in X_t \setminus Y_t$, a later $x \in \Delta_{t'}^{S-}$ removes x from the union $U_{t'}$ but cannot revoke the

Functionality $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$

State: $X := \emptyset, Y := \emptyset$. Each update list splits as $\Delta_t^S = \Delta_t^{S+} \sqcup \Delta_t^{S-}$ and $\Delta_t^R = \Delta_t^{R+} \sqcup \Delta_t^{R-}$.

- **Init.** Sender sends X_0, receiver sends Y_0; set $X := X_0$, $Y := Y_0$; deliver $(|X|, |Y|)$ to sender and $(X \cup Y, |X|, |Y|)$ to receiver.
- **Epoch** $t = 1, \ldots, T$**.** On (Δ_t^S, Δ_t^R): return $\perp$ if any of $\Delta_t^{R+} \cap Y \neq \emptyset$, $\Delta_t^{R-} \not\subseteq Y$, $\Delta_t^{R+} \cap \Delta_t^{R-} \neq \emptyset$ (or sender analogue) hold; else update $X := (X \setminus \Delta_t^{S-}) \cup \Delta_t^{S+}$, $Y := (Y \setminus \Delta_t^{R-}) \cup \Delta_t^{R+}$; deliver $(|X|, |Y|, |\Delta_t^{R+}|, |\Delta_t^{R-}|)$ to sender and $(X \cup Y, |X|, |Y|, |\Delta_t^{S+}|, |\Delta_t^{S-}|)$ to receiver.

Fig. 4: Two-sided-update adaptive-input updatable PSU functionality.

receiver's knowledge. The functionality models this directly: its per-epoch output is the sequence $\{U_t\}_{t=0}^T$, with no notion of un-revealing a past union element. Re-inserting an already-observed x as honest input therefore leaks no new sender information, since the ideal output already contains that knowledge.

4.2 Protocol overview

A one-time Setup encodes both parties' initial sets and builds the union $U_0 = X_0 \cup Y_0$ by running the base PSU protocol once over the full sets. Both parties then keep their sets in uOKVS encodings and update them once per epoch. The per-epoch protocol runs in five phases that play three roles: encoding refresh, local insertion, and membership query. Phase 1 refreshes both encodings under the current epoch tag. Phase 2 inserts the receiver's own new elements locally. Phases 3–5 resolve the remaining update classes, each by a single-item *base-PSU query* that tests one updated element against the other party's encoded set, so the query layer scales with the update rather than the whole set.

The single-item base-PSU query at Phases 3–5 runs in two directions that differ in what they reveal. A *receiver-update* query runs against the sender's active set X_t and a *sender-update* query against the receiver's protected set $P_t^R = Y_t \cup \Delta_t^{R-}$. A receiver-update query reveals a membership bit, whereas a sender-update query reveals an element only when it lies outside P_t^R. This reveal pattern forces a set asymmetry, in which the receiver includes its just-deleted keys Δ_t^{R-} in P_t^R while the sender encodes only X_t. Appendix D shows both choices are necessary. Cleartext-output OPRFs at either side would leak the membership index set as shown in Remark 1, and are ruled out.

4.3 Parameters and setup

Width parameters. The $\mathsf{so\text{-}OPRF}$ output and uOKVS value share a κ-bit length, and the ssPEQT takes κ-bit inputs, so the OPRF output needs no

truncation. Each uOKVS uses an expansion factor ϵ and band width w, with $m_\alpha = \lceil (1+\epsilon) N^\alpha_{\max} \rceil$ for $\alpha \in \{S, R\}$.

Setup. Each party samples its own PRF key locally, k^S for sender and k^R for receiver, and passes it to both its $\mathcal{F}_{\mathsf{so\text{-}OPRF}}.\mathsf{Init}$ and $\mathcal{F}_{\mathsf{uOKVS}}.\mathsf{Init}$. This same-key invariant is what makes each base-PSU query compute the correct membership bit. The initial union $U_0 = X_0 \cup Y_0$ is built by a one-time base-PSU run. The parties invoke $\mathsf{BaseEncode}$, the protocol-level call of $\mathcal{F}_{\mathsf{uOKVS}}.\mathsf{Init}$, to produce (ϕ^S_0, L^S_0) and (ϕ^R_0, L^R_0). The receiver sends L^R_0 to the sender, and both parties run $|X_0|$ base-PSU queries that reveal each $x \in X_0$ to the receiver exactly when $x \notin Y_0$. This is the source of the $|X_0|$ summand in Theorem 5. Correlation material such as VOLE bases, binary-OLE, and random-OT is generated offline and topped up per epoch.

Setup-time vs. open-ended parameters. The Setup phase fixes $N^S_{\max}$, $N^R_{\max}$, w, ϵ, placement seed ρ, and PRF keys k^S, k^R, but not the epoch count T. A single Setup supports an open-ended sequence of epochs, and Theorem 5 establishes its security for any $T = \mathsf{poly}(\lambda)$. Rank-compatibility is checked per execution, at no additional Setup cost. Capacity exhaustion alone triggers a reinitialization with a fresh ρ', independent of T, as described in Section 3.3.

4.4 Per-epoch protocol

Figure 5 specifies the per-epoch protocol $\Pi^{\mathrm{amort}}_{\mathsf{UPSU}}$ in five phases that maintain the invariant $U_t = X_t \cup Y_t$ across epochs. Given $U_{t-1} = X_{t-1} \cup Y_{t-1}$, the new union $U_t = X_t \cup Y_t$ changes only within the per-epoch updates:

$$\underbrace{(U_t \setminus U_{t-1})}_{\text{Addition}} \cup \underbrace{(U_{t-1} \setminus U_t)}_{\text{Deletion}} \subseteq \underbrace{\Delta^{S+}_t \cup \Delta^{R+}_t}_{\text{Addition}} \cup \underbrace{\Delta^{S-}_t \cup \Delta^{R-}_t}_{\text{Deletion}}.$$

Each party holds a uOKVS state from the construction of Section 3.3. In Phase 1, each party advances this state with Update. The new states ϕ^S_t and ϕ^R_t have active sets X_t and Y_t, respectively. The receiver then re-encodes its protected set $P^R_t = Y_t \cup \Delta^{R-}_t$ with values $F_{k^R}(t, y)$ at each $y \in P^R_t$, while the sender re-encodes only its active set X_t with values $F_{k^S}(t, x)$ at each $x \in X_t$. Because each value is bound to the epoch tag t, decoding a fixed key from L^R_t and L^R_{t+1} gives values that agree with probability only $2^{-\kappa}$, even when the key stays in the protected set across both epochs. A key's presence therefore cannot be linked across epochs, even if it persists in the protected set.

Phase 2 adds the receiver's own new elements Δ^{R+}_t to U_t locally. Phases 3–5 resolve the remaining update classes, each with a single-item base-PSU query. Phase 3 handles Δ^{R-}_t, whose elements leave Y_t but stay in U_t if and only if they remain in X_t. A receiver-update query then tests whether $y \in X_t$ and reveals only the membership bit, since the receiver already knows y. Phases 4–5 handle Δ^{S+}_t and Δ^{S-}_t with sender-update queries that test whether $x \in P^R_t$. A membership bit does not suffice here, since the receiver does not know x. A tagged $\mathcal{F}^{\mathrm{shared}}_{\mathsf{OT}}$ instead delivers x to the receiver only when $x \notin P^R_t$, and hides it otherwise. Phase 4 adds revealed $x \in \Delta^{S+}_t$ to U_t, and Phase 5 removes revealed

Protocol $\Pi^{\text{amort}}_{\mathsf{UPSU}}$: amortized updatable PSU (epoch t)

Setup (once). The sender and receiver locally sample PRF keys k^S, k^R and run $\mathsf{BaseEncode}$ to obtain (ϕ_0^S, L_0^S) and (ϕ_0^R, L_0^R). The receiver sends L_0^R to the sender, and both parties run $|X_0|$ base-PSU queries delivering $U_0 = X_0 \cup Y_0$ to the receiver. The parties publicly announce $(|X_0|, |Y_0|)$ and generate correlations offline.
Per-epoch t. Inputs $\Delta_t^S = \Delta_t^{S+} \sqcup \Delta_t^{S-}$ and $\Delta_t^R = \Delta_t^{R+} \sqcup \Delta_t^{R-}$. The parties broadcast their update sizes $(|\Delta_t^{S+}|, |\Delta_t^{S-}|)$ and $(|\Delta_t^{R+}|, |\Delta_t^{R-}|)$, matching the leakage profile $\mathcal{L}(\cdot)$. The receiver sets $U_t \leftarrow U_{t-1}$ in place.

Phase 1 (Encoding refresh).

- The sender computes $\phi_t^S \leftarrow \mathsf{Update}(\phi_{t-1}^S, \Delta_t^S)$ and $L_t^S \leftarrow \mathsf{ReEncode}(\phi_t^S, t)$ on the active set X_t, then sends L_t^S.
- The receiver computes $\phi_t^R \leftarrow \mathsf{Update}(\phi_{t-1}^R, \Delta_t^R)$ and $L_t^R \leftarrow \mathsf{ReEncodeProtected}(\phi_t^R, t, P_t^R)$ on the protected set $P_t^R = Y_t \cup \Delta_t^{R-}$, then sends L_t^R.

Update returns $\perp$ on compatibility violation.
Phase 2 (Δ_t^{R+} local). The receiver adds each $y \in \Delta_t^{R+}$ to U_t with no communication.
Phase 3 (Δ_t^{R-}, receiver-update queries). For each $y \in \Delta_t^{R-}$ in canonical order:

1. The parties invoke $\mathcal{F}_{\mathsf{so\text{-}OPRF}}.\mathsf{eval}(k^S; t, y)$ to obtain (y^0, y^1) with $y^0 \oplus y^1 = F_{k^S}(t, y)$.
2. The receiver decodes $\tilde{u} \leftarrow \mathsf{Decode}(L_t^S, y)$.
3. The parties invoke $\mathcal{F}_{\mathsf{ssPEQT}}$ on $(y^1 \oplus \tilde{u}, y^0)$ to obtain (a^S, a^R) with $a^S \oplus a^R = m_y = \mathbf{1}[y \in X_t]$.
4. The sender opens a^S, and the receiver recovers $m_y = a^S \oplus a^R$ and removes y from U_t when $m_y = 0$.

Phase 4 (Δ_t^{S+}, sender-update queries). For each $x \in \Delta_t^{S+}$ in a fresh π-permuted order:

1. The parties invoke $\mathcal{F}_{\mathsf{so\text{-}OPRF}}.\mathsf{eval}(k^R; t, x)$ to obtain (x^0, x^1) with $x^0 \oplus x^1 = F_{k^R}(t, x)$.
2. The sender decodes $\tilde{v} \leftarrow \mathsf{Decode}(L_t^R, x)$.
3. The parties invoke $\mathcal{F}_{\mathsf{ssPEQT}}$ on $(x^1 \oplus \tilde{v}, x^0)$, obtaining (b^S, b^R) with $b^S \oplus b^R = z_x = \mathbf{1}[x \in P_t^R]$.
4. The parties invoke $\mathcal{F}_{\mathsf{OT}}^{\text{shared}}$: the sender inputs messages $(M_0, M_1) = ((1, x), (0, 0^\kappa))$ and its choice share b^S, and the receiver inputs b^R. The receiver obtains $M_{b^S \oplus b^R} = M_{z_x}$ and reads the tag bit (tag $1 \Leftrightarrow z_x = 0$).
5. If the tag is 1, the receiver adds x to U_t since $x \notin P_t^R$. Otherwise, x is not added.

Phase 5 (Δ_t^{S-}, sender-update queries). For each $x \in \Delta_t^{S-}$ in a fresh π-permuted order, the parties proceed exactly as in Phase 4, except that the receiver removes x from U_t when the tag is 1, since $x \notin P_t^R$.
Output. $U_t = X_t \cup Y_t$.

Fig. 5: Amortized updatable-PSU per-epoch protocol.

$x \in \Delta_t^{S-}$ from U_t. Although Phases 3–5 loop over individual elements, each phase batches its queries into one $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ round and one $\mathcal{F}_{\mathsf{ssPEQT}}$ round over fresh correlation material, and the sender-update phases add one more $\mathcal{F}_{\mathsf{OT}}^{\text{shared}}$ round for the conditional reveal.

4.5 Complexity

Table 2 lists the per-epoch costs in terms of band width w, OPRF output length κ, encoding length $m_\alpha = \lceil (1+\epsilon) N_{\max}^\alpha \rceil$ for $\alpha \in \{S, R\}$, and update size $\Delta_t = |\Delta_t^S| + |\Delta_t^R|$. We evaluate symmetric deployments, so $N_{\max}^S = N_{\max}^R := N_{\max}$, and the refresh cost tracks this capacity through m_α rather than the active size $n = |X_t| + |Y_t|$.

Table 2: Per-epoch asymptotic costs scale with Δ_t in the query layer (Phases 3–5) and with m_S, m_R in the refresh (Phase 1).

Phase	Compute	Online comm.	Offline comm.
1a (Update structural)	$O(\lvert\Delta_t^R\rvert w^2 + \lvert\Delta_t^S\rvert w^2)$	0	—
1b (refresh and transmit)	$O((m_S + m_R)w\kappa)$	$O((m_S + m_R)\kappa)$	—
2 (Δ_t^{R+} local)	$O(\lvert\Delta_t^{R+}\rvert)$	0	—
3 (Δ_t^{R-})	$O(\lvert\Delta_t^{R-}\rvert(w + \kappa))$	$O(\lvert\Delta_t^{R-}\rvert\kappa)$	$O(\lvert\Delta_t^{R-}\rvert\kappa)$
4 (Δ_t^{S+})	$O(\lvert\Delta_t^{S+}\rvert(w + \kappa))$	$O(\lvert\Delta_t^{S+}\rvert\kappa)$	$O(\lvert\Delta_t^{S+}\rvert\kappa)$
5 (Δ_t^{S-})	$O(\lvert\Delta_t^{S-}\rvert(w + \kappa))$	$O(\lvert\Delta_t^{S-}\rvert\kappa)$	$O(\lvert\Delta_t^{S-}\rvert\kappa)$
Total	$\mathbf{O((m_S+m_R)w\kappa + \Delta_t(w^2+w+\kappa))}$	$O((m_S+m_R)\kappa+\Delta_t\kappa)$	$O(\Delta_t\kappa)$

Remark 5 (Bandwidth trade-off). The query layer (Phases 3–5) costs $O(\Delta_t(w+\kappa))$ and scales with the update Δ_t, not with the active set. The encoding refresh (Phase 1) is instead a fixed per-epoch floor that costs $O((m_S + m_R)w\kappa)$ in compute and $O((m_S + m_R)\kappa)$ in bandwidth at every epoch, regardless of Δ_t. This refresh floor is unavoidable. DE forces fresh encoding randomness each epoch, so the encoding cannot be sent as a partial delta.

Cost vs. static full re-encode. Our per-epoch cost splits into a refresh term and an update term,

$$C_{\text{amort}} = \Theta((m_S + m_R)w\kappa) + \Theta(\Delta_t(w^2 + w + \kappa)).$$

The static baseline instead re-encodes the full $N_{\max}$-row band from scratch each epoch, and its cost $C_{\text{static}} = \Theta(N_{\max}w^2)$ is dominated by fresh Gaussian elimination. Per stored row our refresh pays only $w\kappa$ against the static w^2, so C_{amort} stays below C_{static} across the realistic range of Δ_t. The update term overtakes the refresh term only at the internal cross-over $\Delta^* = (m_S + m_R)w\kappa/(w^2 + w + \kappa)$. For our implementation, with $w = 196$, $\kappa = 128$, and $\epsilon = 0.1$, this is about $1.4N_{\max}$, far above any realistic update. The per-epoch cost therefore stays on a refresh-dominated floor that is flat in Δ_t.

4.6 Running example

We illustrate the per-epoch protocol of Figure 5 with a toy example. Let $X_{t-1} = \{c, d\}$ and $Y_{t-1} = \{a, b, c\}$, yielding $U_{t-1} = \{a, b, c, d\}$. At epoch t, the receiver updates $\Delta_t^{R+} = \{e\}, \Delta_t^{R-} = \{c\}$, giving $Y_t = \{a, b, e\}$. The sender updates $\Delta_t^{S+} = \{b\}, \Delta_t^{S-} = \{d\}$, giving $X_t = \{b, c\}$. The expected union is $U_t = \{a, b, c, e\}$. Phase 1 refreshes both encodings $P_t^R = \{a, b, c, e\}$ and $X_t = \{b, c\}$. Phase 2 adds e locally. Phase 3 tests $c \in X_t$. Since $c \in X_t$, the receiver keeps c in U_t. Phases 4–5 test whether $b \in P_t^R$ and $d \in P_t^R$, respectively. Phase 4 adds nothing, since $b \in P_t^R$ and is already in Y_t. Phase 5 removes d, since $d \notin P_t^R$ and was contributed solely by X. The result is $U_t = (\{a, b, c, d\} \cup \{e\}) \setminus \{d\} = \{a, b, c, e\}$.

5 Security Analysis

All theorems in this section presume rank-compatible executions in the sense of Definition 1. Rank failures are public availability aborts outside $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$, as discussed in Remark 3.

We establish correctness in Section 5.2 and specify the single-item base-PSU simulators in Section 5.3, and prove the security in Section 5.4. Then, Section 5.5 abstracts the base-PSU query into a pluggable slot that admits any static OKVS-based single-item PSU realizing this interface.

The security argument runs over two layers. The protocol layer reduces Phases 3–5 to $O(\Delta_t)$ base-PSU queries over fresh, independent correlation material, so adaptive updates do not amplify the bound. The encoding layer goes through the $\mathcal{F}_{\mathsf{uOKVS}}$ hybrid, which idealizes DE but *not* map-hiding. The PRF-indexed map-hiding therefore stays computational and is charged through Lemma 3. The precise per-query and per-realization terms are collected in Theorem 5 and Corollary 1.

5.1 Setup of the reduction

Definition 3 (Compatible execution). *An execution is* compatible *if, at each epoch t, the sender's and receiver's inputs are finite sets over the global key space, the update sequences Δ_t^S, Δ_t^R respect the active/dead bookkeeping invariant of uBand-OKVS—no duplicate inserts on active keys, no deletes on non-stored keys, and no key both inserted and deleted within the same epoch—and neither party's new-key capacity is exceeded.*

The persistent state consists of PRF keys k^S, k^R, uOKVS states ϕ^S, ϕ^R, and the receiver cache U_t. Offline correlation material is partitioned per epoch.

The base-PSU query of Section 4.2, underlying the Setup and Phases 3–5, realizes one membership test against an OKVS-encoded set in the $(\mathcal{F}_{\mathsf{so\text{-}OPRF}}, \mathcal{F}_{\mathsf{OT}}^{\text{shared}}, \mathcal{F}_{\mathsf{ssPEQT}})$-hybrid. A receiver-update query maps (y, X_t) to $\mathbf{1}[y \in X_t]$, and a sender-update query maps (x, P_t^R) to $\mathbf{1}[x \in P_t^R]$. Inside the hybrid each query simulator incurs only the ideal-world slack $\mathsf{Adv}_{\text{ideal}}^{\text{base}} := \mathsf{negl}(\lambda)$ from Band-OKVS random-decoding collision and dummy-key residuals. The full simulator specification appears in Proposition 1.

5.2 Per-epoch correctness

We write ε_{rd} for an upper bound on the random-decoding probability

$$\Pr[\mathsf{Decode}(\mathsf{pp}, \mathsf{Encode}(\mathsf{pp}, I), k^*) = v^*]$$

at any non-stored key $k^* \notin \mathsf{supp}(I)$ and any $v^* \in V$, where the values of I are the PRF values $F_k(t, x)$ used by our protocol.

Assumption 3 (Random-decoding bound for Band-OKVS). *Band-OKVS satisfies $\varepsilon_{\text{rd}} \le 2^{-\lambda}$ for the statistical security parameter λ.*

This bound is justified by the standard Band-OKVS oblivious-decode analysis of [7] applied to the PRF-indexed output distribution. Because the bound is statistical, it subsumes the computational requirement of Section 2.1. The PRF-to-random-function hop of Lemma 3 reduces this distribution to the uniform-value case in which the lemma of [7] applies, with the residual $\mathsf{Adv}_F^{\mathsf{PRF}}$ slack already charged elsewhere.

Theorem 4 (Correctness of the amortized UPSU). *For any compatible execution of $\Pi_{\mathsf{UPSU}}^{\mathrm{amort}}$ in the sense of Definition 3 and any fixed epoch $t \geq 1$, if the receiver's cache satisfies $U_{t-1} = X_{t-1} \cup Y_{t-1}$ at the start of epoch t, then the cache produced at the end of the epoch satisfies $U_t = X_t \cup Y_t$, except with probability at most*

$$\Delta_{\max} \cdot \varepsilon_{\mathrm{rd}} + \mathsf{negl}(\lambda).$$

The first term aggregates the Band-OKVS false-positive contribution across at most $|\Delta_t^{S+}| + |\Delta_t^{S-}| + |\Delta_t^{R-}| \leq \Delta_{\max}$ base-PSU queries in this epoch, each of which decodes L_t^S or L_t^R at a non-stored key. The negligible term absorbs dummy-key collisions and static $\mathsf{Encode}/\mathsf{Decode}$ correctness slack, while rank-compatibility in the sense of Definition 1 eliminates the $\mathcal{F}_{\mathsf{uOKVS}}$ abort branch. Across T epochs, the union bound gives $\Pr[\exists t \leq T : U_t \neq X_t \cup Y_t] \leq T \cdot \Delta_{\max} \cdot \varepsilon_{\mathrm{rd}} + T \cdot \mathsf{negl}(\lambda)$.

Proof (Proof sketch). We proceed by induction on t. The base case $t = 0$ holds, since the Setup establishes $U_0 = X_0 \cup Y_0$. Assuming $U_{t-1} = X_{t-1} \cup Y_{t-1}$, Phase 1 refreshes the encodings without changing U_t, and Phase 2 adds Δ_t^{R+} to U_t unconditionally. The remaining work falls to Phases 3–5.

Phase 3. For each $y \in \Delta_t^{R-}$, Phase 3 computes $m_y = \mathbf{1}[y \in X_t]$ and removes y from U_t when $m_y = 0$. Since $y \notin Y_t$ after Update, $y \in X_t \cup Y_t$ holds exactly when $y \in X_t$. Therefore, Phase 3 correctly removes y at $m_y = 0$, while the sender's contribution keeps y in the union at $m_y = 1$.

Phase 4. For $x \in \Delta_t^{S+}$, Phase 4 queries L_t^R to obtain $z_x = \mathbf{1}[x \in P_t^R]$ and appends x to U_t when $z_x = 0$. Since $x \in X_t$, the element x must lie in U_t. The append at $z_x = 0$ achieves this directly, while at $z_x = 1$ the receiver's prior contribution keeps x in U_t. This contribution comes from either $x \in Y_t$ via cache carry-over or $x \in \Delta_t^{R-}$ via Phase 3, which keeps x since $m_x = 1$.

Phase 5. For $x \in \Delta_t^{S-}$, Phase 5 queries L_t^R to obtain $z_x = \mathbf{1}[x \in P_t^R]$ and removes x from U_t when $z_x = 0$. Since $x \notin X_t$, the element x lies in U_t if and only if $x \in Y_t$. The two cases of z_x then mirror Phase 4 with append replaced by removal, except that Phase 3 now *removes* $x \in \Delta_t^{R-}$ since $m_x = 0$.

The full case analysis, including the overlaps $\Delta_t^{S+} \cap \Delta_t^{R-}$ and $\Delta_t^{S-} \cap \Delta_t^{R-}$, appears in Appendix E.

5.3 Single-item base-PSU simulators

Proposition 1 (Base-PSU query simulator). *For every direction $\delta \in \{\mathsf{su}, \mathsf{ru}\}$, with su the sender-update direction and ru the receiver-update direction,*

and corruption side $C \in \{S, R\}$, there is a PPT simulator $\mathcal{S}_C^{\delta}$ for one base-PSU query in the $(\mathcal{F}_{\mathsf{so\text{-}OPRF}}, \mathcal{F}_{\mathsf{OT}}^{\text{shared}}, \mathcal{F}_{\mathsf{ssPEQT}})$-hybrid model, with distinguishing advantage $\mathsf{Adv}_{\text{ideal}}^{\text{base}} = \mathsf{negl}(\lambda)$.

The simulator inputs follow the corruption-side leakage profile of $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$. A receiver-corrupted simulator consumes the ideal output bit, namely z_x for the su direction and m_y for the ru direction. A sender-corrupted simulator consumes no bit, since the sender's view is uniform shares from $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$, $\mathcal{F}_{\mathsf{ssPEQT}}$, and $\mathcal{F}_{\mathsf{OT}}^{\text{shared}}$. Full inputs and transcripts for each corruption side and direction appear in Appendix F.

Proof (Proof sketch). The two directions are symmetric, hence we treat $\delta = \mathsf{su}$ and obtain the ru case by role-swap.

Sender-corrupted. The simulator $\mathcal{S}_S^{\mathsf{su}}$ samples $x^1 \leftarrow V$, which is perfect. It then invokes the $\mathcal{F}_{\mathsf{ssPEQT}}$ simulator on a uniform sender share and runs the $\mathcal{F}_{\mathsf{OT}}^{\text{shared}}$ simulator with the sender's own payload pair $((1, x), (0, 0^{\kappa}))$. The tagged payload is placed in the slot determined by the ssPEQT output share b^S. Since $\mathcal{F}_{\mathsf{ssPEQT}}$ delivers additive shares of z_x, the share b^S is uniform and independent of z_x, so this placement leaks no information about z_x. The dummy history L_t^R from $\mathcal{F}_{\mathsf{uOKVS}}$ depends on P_t^R only through the already-leaked sizes.

Receiver-corrupted. For a whole sender-update batch, the simulator uses the receiver's ideal outputs to derive the payloads that should be opened. Phase 4 opens $(U_t \setminus U_{t-1}) \setminus \Delta_t^{R+}$ and Phase 5 opens $(U_{t-1} \setminus U_t) \setminus \Delta_t^{R-}$. The simulator samples the positions of these payloads uniformly within the leaked batch length, matching the sender's fresh permutation π. All remaining positions have $z_x = 1$ and reveal only the sentinel. For each slot, the simulator invokes the $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ simulator on a uniform receiver share x^0 and the $\mathcal{F}_{\mathsf{ssPEQT}}$ simulator with forced output z_x. The residual error $\mathsf{negl}(\lambda)$ comes from dummy collisions.

The receiver-update case ($\delta = \mathsf{ru}$) follows by role-swap, as noted above.

5.4 Main theorem and proof

Theorem 5 (Protocol security in the $\mathcal{F}_{\mathsf{uOKVS}}$-hybrid). *$\Pi_{\mathsf{UPSU}}^{\text{amort}}$ realizes $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$ in the $(\mathcal{F}_{\mathsf{uOKVS}}, \mathcal{F}_{\mathsf{so\text{-}OPRF}}, \mathcal{F}_{\mathsf{OT}}^{\text{shared}}, \mathcal{F}_{\mathsf{ssPEQT}})$-hybrid model under adaptive-input semi-honest corruption, for any $T = \mathsf{poly}(\lambda)$ epochs of compatible execution. For each corruption side $C \in \{S, R\}$, there exists a PPT simulator $\mathcal{S}_C$, constructed independently per side. For every PPT adversary $\mathcal{A}_C$ and PPT adaptive-input environment $\mathcal{Z}$,*

$$\mathsf{EXEC}_{\Pi_{\mathsf{UPSU}}^{\text{amort}}, \mathcal{A}_C, \mathcal{Z}} \overset{c}{\equiv} \mathsf{IDEAL}_{\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}, \mathcal{S}_C, \mathcal{Z}}$$

with distinguishing advantage at most

$$(|X_0| + T \cdot \Delta_{\max}) \cdot \mathsf{Adv}_{\text{ideal}}^{\text{base}} + \mathsf{Adv}_{T+1}^{\mathcal{F}_{\mathsf{uOKVS}}} + \mathsf{negl}(\lambda).$$

The first summand $\mathsf{Adv}_{\text{ideal}}^{\text{base}} = \mathsf{negl}(\lambda)$ is the per-query simulator error of Proposition 1, multiplied by the total of $|X_0| + T \cdot \Delta_{\max}$ base-PSU queries from the

Setup and the T epochs. The second summand $\mathsf{Adv}^{\mathcal{F}_{\mathsf{uOKVS}}}_{T+1}$ equals the advantage of the stateful adaptive hiding game of Definition 2 and charges the honest-side map-hiding substitution.

Corollary 1. *Let $\Pi^{\mathrm{real}}_{\mathsf{UPSU}}$ be the protocol obtained from Section 4.4 by instantiating its four ideal functionalities as follows: $\mathcal{F}_{\mathsf{uOKVS}}$ by uBand-OKVS of Section 3.3, applied to both parties; $\mathcal{F}^{\mathrm{shared}}_{\mathsf{OT}}$ by the swap-and-forward construction of Remark 2; $\mathcal{F}_{\mathsf{ssPEQT}}$ by ssPEQT on preprocessed binary-OLE; and $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ by so-OPRF of Section 6. Each party follows the PRF-key setup of Section 4.3. Let $N_{\mathsf{ssPEQT}} \le |X_0| + T \cdot \Delta_{\max}$ be the total number of ssPEQT invocations across every base-PSU query, and let $N_{\mathsf{OT}} \le |X_0| + \sum_t (|\Delta_t^{S+}| + |\Delta_t^{S-}|)$ be the total number of $\mathcal{F}^{\mathrm{shared}}_{\mathsf{OT}}$ invocations across sender-update queries. The real-world distinguishing advantage is at most*

$$\underbrace{(|X_0| + T \cdot \Delta_{\max}) \cdot \mathsf{Adv}^{\mathrm{base}}_{\mathrm{ideal}} + \mathsf{Adv}^{\mathcal{F}_{\mathsf{uOKVS}}}_{T+1}}_{\textit{hybrid bound (Theorem 5)}}$$
$$+ \underbrace{\mathsf{Adv}^{\mathcal{F}_{\mathsf{uOKVS}}}_{T+1}}_{\textit{corrupted-side } \mathcal{F}_{\mathsf{uOKVS}} \textit{ realization}} + \underbrace{\mathsf{Adv}^{\mathsf{so\text{-}OPRF}}_{T_{\mathsf{su}}} + \mathsf{Adv}^{\mathsf{so\text{-}OPRF}}_{T_{\mathsf{ru}}}}_{\textit{two } \mathcal{F}_{\mathsf{so\text{-}OPRF}} \textit{ realizations}}$$
$$+ N_{\mathsf{ssPEQT}} \cdot \mathsf{Adv}^{\mathsf{ssPEQT}} + N_{\mathsf{OT}} \cdot \mathsf{Adv}^{\mathsf{OT}} + \mathsf{negl}(\lambda),$$

where $T_{\mathsf{su}} \le |X_0| + T \cdot \Delta_{\max}$ and $T_{\mathsf{ru}} \le T \cdot \Delta_{\max}$ count the sender-update and receiver-update OPRF invocations.

Proof (Proof sketch). For each corruption side $C \in \{S, R\}$, the simulator $\mathcal{S}_C$ operates in the hybrid model with direct access to $\mathcal{F}_{\mathsf{uOKVS}}$, $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$, $\mathcal{F}^{\mathrm{shared}}_{\mathsf{OT}}$, and $\mathcal{F}_{\mathsf{ssPEQT}}$. It receives the corrupted party's update list together with the corruption-side leakage of $\mathcal{F}^{\text{2-update}}_{\mathsf{UPSU}}$. We construct a hybrid chain H_0, H_1, H_2, H_3, H_4 connecting the real and ideal executions, and we list the advantage of each hop alongside.

H_0: This is the real execution.

H_1: In this hybrid, we replace each of the Setup's $|X_0|$ sender-update queries with the simulator of Proposition 1. The simulator delivers $U_0 = X_0 \cup Y_0$ to the corrupted receiver, or nothing to the corrupted sender. The bookkeeping of (L_0^S, L_0^R) remains real. By Proposition 1, the hop advantage is $|X_0| \cdot \mathsf{Adv}^{\mathrm{base}}_{\mathrm{ideal}}$.

H_2: In this hybrid, we replace each per-epoch base-PSU query with the corresponding direction-specific simulator of Proposition 1. Each epoch issues at most $\Delta_{\max}$ such queries, so over T epochs the hop advantage is $T \cdot \Delta_{\max} \cdot \mathsf{Adv}^{\mathrm{base}}_{\mathrm{ideal}}$.

H_3: In this hybrid, we replace the honest party's $T+1$ $\mathcal{F}_{\mathsf{uOKVS}}$ epoch encodings, namely $\{L_\tau^{\bar{C}}\}_{\tau=0}^{T}$, with an independent dummy execution driven by a size-matched update list chosen by $\mathcal{S}_C$. Here, $\tau = 0$ is the Init/BaseEncode output and $\tau \ge 1$ are refresh outputs. The substitution is exactly the stateful adaptive hiding game of Definition 2, bounded by Lemma 3. The hop advantage is $\mathsf{Adv}^{\mathcal{F}_{\mathsf{uOKVS}}}_{T+1}$. The auxiliary view of the environment, which contains ideal $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$, $\mathcal{F}_{\mathsf{ssPEQT}}$, and $\mathcal{F}^{\mathrm{shared}}_{\mathsf{OT}}$ shares, is rewired consistently with the dummy map.

H_4: In this hybrid, we replace the leftover honest-party cleartext-input wiring, such as PRF evaluations and Decode calls, with the ideal output and leakage that $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$ provides to the corrupted side. This is structural bookkeeping inside $\mathcal{S}_C$ and is perfect given the previous hops.

The hop advantages, together with a residual $\mathsf{negl}(\lambda)$ from dummy-key collisions and static $\mathsf{Encode}/\mathsf{Decode}$ slack, sum to the bound of Theorem 5, and Corollary 1 follows by replacing each ideal functionality with its realization. The full proof, with the case split over $C \in \{S, R\}$, adaptive-update bookkeeping, and rank-compatibility side-condition, appears in Appendix G.

5.5 Framework: pluggable base queries

Theorem 5 and Corollary 1 are parametric in Proposition 1. The encoding-layer hop enters the bound once, through $\mathsf{Adv}_{T+1}^{\mathcal{F}_{\mathsf{uOKVS}}}$, and is independent of the per-query machinery. Abstracting this machinery yields a lifting theorem for static OKVS-based single-item PSU.

Definition 4 (Base-PSU query). *A* base-PSU query *is a PPT two-party subprotocol $\mathcal{Q}$ in the $(\mathcal{F}_{\mathsf{so\text{-}OPRF}}, \mathcal{F}_{\mathsf{OT}}^{\text{shared}}, \mathcal{F}_{\mathsf{ssPEQT}})$-hybrid with the following interface. On a sender-update query, the holder inputs (k, L^R), where L^R is the current $\mathcal{F}_{\mathsf{uOKVS}}$ output and k the PRF key, and the query party inputs a single key x. Both parties obtain XOR-shares of $\mathbf{1}[x \in \mathsf{active}(L^R)]$ except with probability $\epsilon_{\mathcal{Q}}^{\text{corr}} = \mathsf{negl}(\lambda)$. The corruption-side leakage is (Y_t, z_x) for receiver-corruption and $|Y_t|$ for sender-corruption. Privacy holds via a PPT simulator $\mathcal{S}_{\mathcal{Q}}$ that takes this leakage and produces transcripts at distinguishing advantage $\mathsf{Adv}_{\text{query}}^{\mathcal{Q}} = \mathsf{negl}(\lambda)$. Receiver-update queries swap the roles symmetrically.*

Corollary 2 (Lifting). *Let $\mathcal{Q}$ be a base-PSU query, and let $\mathsf{Lift}(\mathcal{Q})$ be the protocol obtained from Section 4.4 by instantiating Phase 3 with receiver-update $\mathcal{Q}$, Phases 4–5 with sender-update $\mathcal{Q}$, and the Setup with $|X_0|$ sender-update invocations of $\mathcal{Q}$. $\mathsf{Lift}(\mathcal{Q})$ realizes $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$ in the $(\mathcal{F}_{\mathsf{uOKVS}}, \mathcal{F}_{\mathsf{so\text{-}OPRF}}, \mathcal{F}_{\mathsf{OT}}^{\text{shared}}, \mathcal{F}_{\mathsf{ssPEQT}})$-hybrid model under adaptive-input semi-honest corruption, with distinguishing advantage at most*

$$(|X_0| + T \cdot \Delta_{\max}) \cdot \mathsf{Adv}_{\text{query}}^{\mathcal{Q}} + \mathsf{Adv}_{T+1}^{\mathcal{F}_{\mathsf{uOKVS}}} + \mathsf{negl}(\lambda).$$

Proof. The hybrid chain is that of Theorem 5, with the per-query simulator slot filled by $\mathcal{S}_{\mathcal{Q}}$ in place of the concrete simulator of Proposition 1.

Query layer. Each base-PSU query is replaced by $\mathcal{S}_{\mathcal{Q}}$ at cost $\mathsf{Adv}_{\text{query}}^{\mathcal{Q}}$, and $\mathsf{Lift}(\mathcal{Q})$ issues $|X_0|$ such queries in the Setup and at most $\Delta_{\max}$ per epoch across T epochs, so a union bound over the $|X_0| + T \cdot \Delta_{\max}$ queries contributes $(|X_0| + T \cdot \Delta_{\max}) \cdot \mathsf{Adv}_{\text{query}}^{\mathcal{Q}}$. Because $\mathcal{Q}$ runs in the $(\mathcal{F}_{\mathsf{uOKVS}}, \mathcal{F}_{\mathsf{so\text{-}OPRF}}, \mathcal{F}_{\mathsf{OT}}^{\text{shared}}, \mathcal{F}_{\mathsf{ssPEQT}})$-hybrid, these functionalities are ideal here and contribute no advantage term. Their realization cost, including $\mathsf{Adv}^{\mathsf{OT}}$, is charged once in Corollary 1.

Encoding layer. The encoding-layer hop is unchanged from Theorem 5 and stays $\mathcal{Q}$-independent. Rank- and update-compatibility are preserved along the chain, so $\mathsf{BaseEncode}/\mathsf{Update}$ and the ideal functionality never take the $\perp$ branch.

Summing the two layers and the residual $\mathsf{negl}(\lambda)$ gives the stated bound.

Table 3: Median per-epoch online wall-time (ms) and communication (MB) over 10 single-threaded runs. Our protocol runs at $T = 10$ and $\Delta = 55$, while baselines re-run static PSU from scratch each epoch.

n	Protocol	Setup (s)	LAN (ms)	WAN (ms)	Comm (MB)	Speedup (LAN)
$n = 2^{16}$	Ours	0.2	391	2630	2.98	–
	JSZG24 [20]	–	4232	21122	137.42	10.8×
	TBZ$^+$25 [30]	–	9232	12383	16.51	23.6×
	KLS26 [21]	–	3540	7781	13.38	9.1×
$n = 2^{18}$	Ours	0.6	587	3618	9.58	–
	JSZG24 [20]	–	11704	65861	577.73	19.9×
	TBZ$^+$25 [30]	–	35037	42412	64.42	59.7×
	KLS26 [21]	–	15353	23166	49.92	26.2×
$n = 2^{20}$	Ours	2.7	1408	6901	35.98	–
	JSZG24 [20]	–	43554	258318	2430.47	30.9×
	TBZ$^+$25 [30]	–	139292	162306	258.75	98.9×
	KLS26 [21]	–	69181	87858	199.22	49.1×

Plug-in catalog. Two instantiations fit Definition 4 and are detailed in Section 6.1: the so-OPRF realization of Proposition 1 and an AltMod-free PKE-based mqRPMT alternative. Future single-item base queries, such as VOLE-based or lattice-based private membership tests, plug into the same interface without re-proving the adaptive-input reduction and inherit the adaptive-hiding guarantee of $\mathcal{F}_{\mathsf{uOKVS}}$.

The framework applies to single-item membership with OKVS-structured encoding. Batch queries such as the cuckoo-based design of KLS26 [21] and circuit-PSI-style protocols lie outside the interface of $\mathcal{Q}$ and require separate analysis. Until a candidate beyond the so-OPRF base is benchmarked, the framework claim rests on the proof that the protocol template factors cleanly through Definition 4, not on empirical evidence.

6 Instantiations and Evaluation

The protocol is stated abstractly over the ideal functionalities of Theorem 5. This section gives two concrete realizations of the base-PSU query and the experimental evaluation.

6.1 Concrete realizations of the base-PSU query

The base-PSU query of Section 4.2 resolves one membership test against an OKVS-encoded set. Every realization Decodes the encoding at the queried key, compares the result against a freshly derived value under ssPEQT to obtain shares of the membership bit, and reveals an element only through the tagged $\mathcal{F}_{\mathsf{OT}}^{\text{shared}}$ conditional opening. The pluggable part is the primitive that supplies

the value to compare, and Section 5.5 admits any static OKVS-based single-item PSU that fills this slot. We give two realizations.

(i) Through $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ (YHW$^+$26 [31]). The value to compare comes from the so-OPRF of Figure 1: one party holds the key k, the other supplies the input, and both obtain uniform shares of $F_k(\cdot)$, which ssPEQT then matches against the decoded value to yield the membership bit. We instantiate the so-OPRF with YHW$^+$26, built from the AltMod weighted PRF [2] over small-field OLE, and this is the realization we implement and benchmark. Its advantage $\mathsf{Adv}_T^{\mathsf{so\text{-}OPRF}}$ appears only in Corollary 1, since the hybrid-model statement of Theorem 5 abstracts $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ and is discharged by the simulation theorem of YHW$^+$26.

(ii) AltMod-free alternative: PKE-based mqRPMT (ZCL$^+$23 [32]). For deployments that prefer to avoid the AltMod assumption, the PKE-based mqRPMT of ZCL$^+$23 realizes the base-PSU query over the same OKVS encoding, with a re-randomizable PKE in place of $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$. Its security follows from the mqRPMT analysis of ZCL$^+$23, and the uOKVS layer and Theorem 5 are unchanged. We do not implement this candidate, and estimate its per-query cost at 5–8× the so-OPRF base query based on the exponentiation counts of ZCL$^+$23.

6.2 Experimental evaluation

Implementation. We implement our protocol in C++.[3] We instantiate $\mathcal{F}_{\mathsf{uOKVS}}$ as the Band-OKVS-based `UpdatableBandOkvs`, $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ as `SoOPRF` [31], and $\mathcal{F}_{\mathsf{ssPEQT}}$ as `ssPEQT`, a secret-shared private equality test on preprocessed binary-OLE. We use the parameter set from the `AltModPrf` reference implementation in secureJoin [2], with $\mathsf{KeySize} = 512$, $\mathsf{MidSize} = 256$, and $\mathsf{OutSize} = \kappa = 128$, and public random matrices $\mathbf{B} \in \mathbb{F}_2^{128\times128}$ and $\mathbf{A} \in \mathbb{F}_3^{256\times128}$ derived from its deterministic seeds. We set the statistical security parameter $\lambda = 40$. We invoke `SoOPRF` in both directions via role-swap, and patch its reference implementation, which hardcodes the AltMod key, to perform the per-bit key-OT setup. Reported numbers therefore include the one-time per-deployment setup.

Benchmarks. We run all experiments on a virtual machine with 4 cores and 8 threads, running Ubuntu 20.04 LTS, hosted on an Intel Xeon Silver 4215 CPU at 3.2 GHz with 128 GB of RAM under Windows Server 2019. All measurements use a single pinned core. Since no prior PSU targets adaptive-input security, we benchmark against the static single-shot protocols JSZG24 [20], TBZ$^+$25 [30], and KLS26 [21], which we re-run single-threaded from scratch each epoch on the same hardware for a matched comparison.

We emulate two network conditions on the loopback interface using `tc-netem`: LAN (10 Gbps, 0.2 ms RTT) and WAN (100 Mbps, 80 ms RTT). Offline (correlations) and online (Phases 1–5) costs are reported separately. Each timing point is the median of 10 independent runs. At every (n, T, Δ) point, the reported U_t from the receiver is byte-verified against a cleartext reference, and we accept only all-epoch verifying runs.

[3] The artifacts related to this work are available at `https://github.com/SeongbongChoi/AIS-uPSU`.

Main results. Table 3 reports the median per-epoch wall-time and speedups over the static baselines. Appendix A plots how this time scales with the update size Δ and the session length T and gives the break-even analysis. At our headline point $(n, T, \Delta) = (2^{20}, 10, 55)$, the protocol completes each epoch in 1408 ms on LAN, 30.9× to 98.9× faster than the three baselines. The same comparison at $n = 2^{18}$ gives 587 ms per epoch and speedups of 19.9× to 59.7×. The concurrent updatable PSI of LMR$^+$26 [24] realizes a different functionality, so a wall-clock comparison does not apply.

Our single-session LAN total of 4108 ms, the one-time 2.7 s Setup plus a single 1408 ms epoch, already beats every baseline. At $n = 2^{20}$, encoding refresh in Phase 1 takes ≈ 1079 ms while the query layer in Phases 3–5 takes ≈ 332 ms, so refresh dominates and the query layer scales linearly in Δ.

WAN bottleneck. On WAN, the per-epoch wall-clock time climbs to 6901 ms at $n = 2^{20}$, a 4.9× blowup over LAN. The bottleneck is the Phase 1 transmission of L_t^R, L_t^S, which saturates the bandwidth at ≈ 36.0 MB combined. Queries contribute under 100 KB per epoch and ride on the already-open TCP window. The refresh-cost structure is inherent to distributional erasure, as discussed in Remark 5. A partial-delta encoding could reduce this to $O(\Delta w^2)$ but would violate DE and reopen the deletion-trace channel of Theorem 1. Deployments tolerant of higher update sizes can raise Δ without penalty, since the query layer is sub-dominant. Alternatively, the band width w can be reduced for lower refresh communication, at the cost of mild rank-failure slack.

References

1. A. Agarwal, D. Cash, M. George, S. Kamara, T. Moataz, and J. Singh. Updatable private set intersection from structured encryption. *IACR Communications in Cryptology*, 2(4):25, 2026.
2. N. Alamati, G. Policharla, S. Raghuraman, and P. Rindal. Improved alternating-moduli prfs and post-quantum signatures. In *CRYPTO 2024*, Lecture Notes in Computer Science, pages 274–308, 2024.
3. F. Alborch, T. Chauvier, A. Faonio, A. Fontaine, F. Karakoç, A. Küpçü, C. Malek, and M. Önen. Updatable private set intersection and beyond: Efficient constructions via circuit private set intersection. Cryptology ePrint Archive, Paper 2025/2147, 2025.
4. S. Badrinarayanan, P. Miao, X. Shi, M. Tromanhauser, and R. Zeng. Updatable private set intersection revisited: Extended functionalities, deletion, and worst-case complexity. In *ASIACRYPT 2024*, Lecture Notes in Computer Science, pages 200–233, 2024.
5. S. Badrinarayanan, P. Miao, and T. Xie. Updatable private set intersection. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2022(2):378–406, 2022.
6. D. Beaver. Precomputing oblivious transfer. In *CRYPTO 1995*, Lecture Notes in Computer Science, pages 97–109, 1995.
7. A. Bienstock, S. Patel, J. Y. Seo, and K. Yeo. Near-optimal oblivious key-value stores for efficient PSI, PSU and volume-hiding multi-maps. In *USENIX Security 2023*, pages 301–318, 2023.

8. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019*, Lecture Notes in Computer Science, pages 489–518, 2019.
9. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE FOCS 2001*, pages 136–145, 2001.
10. Y. Chen, M. Zhang, C. Zhang, M. Dong, and W. Liu. Private set operations from multi-query reverse private membership test. In *PKC 2024*, Lecture Notes in Computer Science, pages 387–416, 2024.
11. A. Davidson and C. Cid. An efficient toolkit for computing private set operations. In *ACISP 2017*, volume 10343 of *Lecture Notes in Computer Science*, pages 261–278, 2017.
12. J. Dumas, A. Galan, B. Grenet, A. Maignan, and D. S. Roche. Optimal communication unbalanced private set union. In *ACNS 2025*, Lecture Notes in Computer Science, pages 107–135, 2025.
13. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.
14. K. B. Frikken. Privacy-preserving set union. In *ACNS 2007*, Lecture Notes in Computer Science, pages 237–252, 2007.
15. J. Gao, S. Nguyen, M. Blanton, and N. Trieu. PULSE: Parallel private set union for large-scale entities. In *ACM CCS 2025*, pages 1784–1798, 2025.
16. J. Gao, S. Nguyen, and N. Trieu. Toward a practical multi-party private set union. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2024(4):622–635, 2024.
17. G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO 2021*, Lecture Notes in Computer Science, pages 395–425, 2021.
18. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, Lecture Notes in Computer Science, pages 145–161, 2003.
19. Y. Jia, S. Sun, H. Zhou, J. Du, and D. Gu. Shuffle-based private set union: Faster and more secure. In *USENIX Security 2022*, pages 2947–2964, 2022.
20. Y. Jia, S. Sun, H. Zhou, and D. Gu. Scalable private set union, with stronger security. In *USENIX Security 2024*, 2024.
21. J. Kim, H. T. Lee, and Y. Son. Private set operations from circuit-based PSI. In *SAC 2026*, pages 1857–1866, 2026.
22. V. Kolesnikov, M. Rosulek, N. Trieu, and X. Wang. Scalable private set union from symmetric-key techniques. In *ASIACRYPT 2019*, Lecture Notes in Computer Science, pages 636–666, 2019.
23. G. Ling, P. Tang, S. Sun, and W. Qiu. Efficient updatable PSI from asymmetric PSI and PSU. *IEEE Transactions on Information Forensics and Security*, 21:2626–2640, 2026.
24. J. Liu, P. Miao, M. Rosulek, X. Shi, and J. Wang. Updatable private set intersection from symmetric-key techniques. In *EUROCRYPT 2026*, Lecture Notes in Computer Science, pages 376–407, 2026.
25. B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. PSI from PaXoS: Fast, malicious private set intersection. In *EUROCRYPT 2020*, Lecture Notes in Computer Science, pages 739–767, 2020.
26. L. Piske and N. Trieu. Is PSI really faster than PSU? achieving efficient PSU with invertible bloom filters. In *EUROCRYPT 2026*, Lecture Notes in Computer Science, pages 438–467, 2026.
27. S. Pu, J. Gao, and N. Trieu. Malicious private set union with two-sided output. In *EUROCRYPT 2026*, Lecture Notes in Computer Science, pages 408–437, 2026.

28. M. O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981. Available at https://eprint.iacr.org/2005/187.
29. P. Rindal and P. Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In *EUROCRYPT 2021*, Lecture Notes in Computer Science, pages 901–930, 2021.
30. B. Tu, Y. Bai, C. Zhang, Y. Cao, and Y. Chen. Fast enhanced private set union in the balanced and unbalanced scenarios. In *USENIX Security 2025*, pages 3437–3456, 2025.
31. X. Yang, M. Hao, C. Weng, R. H. Deng, Y. Wen, and T. Zhang. Efficient fuzzy private set intersection from secret-shared OPRF. In *IEEE Symposium on Security and Privacy*, 2026. Available at https://arxiv.org/abs/2604.14909.
32. C. Zhang, Y. Chen, W. Liu, M. Zhang, and D. Lin. Linear private set union from multi-query reverse private membership test. In *USENIX Security 2023*, pages 337–354, 2023.

A Additional performance measurements

This appendix gives the break-even plots of Figure 6 and the detailed scaling sweeps over (n, T, Δ) behind Table 3.

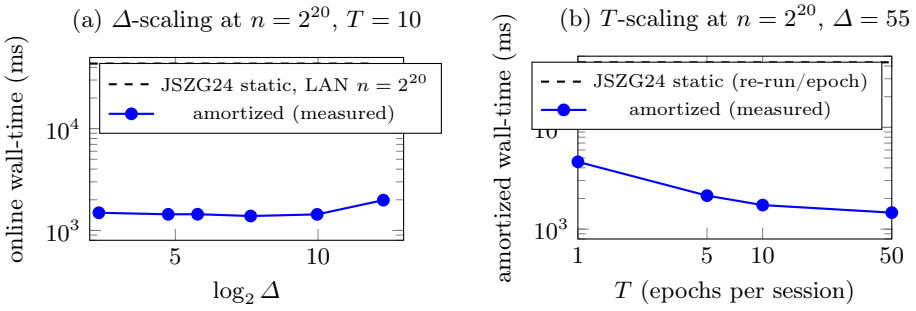

Fig. 6: **(a)** Per-epoch online wall-time vs. update size Δ at $n = 2^{20}$ (data in Table 4). **(b)** Per-epoch wall-time including the amortized one-time Setup vs. the number of epochs T at $n = 2^{20}$ (data in Table 5).

Δ-scaling. Table 4 traces the wall at $n = 2^{20}$ and $T = 10$, and Figure 6(a) plots the same Δ-sweep. The per-epoch wall-clock time stays near a refresh-dominated floor of ≈ 1408 ms across $\Delta \in [5, 1000]$ and the query layer becomes visible only at $\Delta \approx 5000$, far below the asymptotic crossover $\Delta^* \approx 1.4 N_{\max}$ of Section 4.5. Across this sweep, our protocol runs 22–31× faster than the fastest static baseline JSZG24, which is the tightest of the three comparisons.

T-scaling. Table 5 lists the T-sweep at $n = 2^{20}$ and $\Delta = 55$, and Figure 6(b) plots how the one-time Setup amortizes over T. The Setup contributes 270 ms

Table 4: Δ-scaling at $n = 2^{20}$ and $T = 10$.

Δ	Wall/ep (ms)	Phase 1 (ms)	Phases 3–5 (ms)	Comm./ep (MB)
5	1491	1088	309	35.97
27	1440	1068	292	35.98
55	1443	1071	293	35.98
200	1385	1021	285	35.98
1000	1437	1007	348	36.45
5000	1984	1138	719	39.85

per epoch at $T = 10$ and 54 ms per epoch at $T = 50$, after which the per-epoch time converges to a steady state of ≈ 1408 ms. Substituting $C_{\text{Setup}} = 2718$ ms, $C_{\text{amort}} = 1408$ ms, and $C_{\text{static}} \in \{43554, 69181, 139292\}$ ms into $T^* = \lceil C_{\text{Setup}}/(C_{\text{static}} - C_{\text{amort}}) \rceil$ gives $T^* = 1$ in all three comparisons.

Table 5: T-scaling at $n = 2^{20}$ and $\Delta = 55$.

T	Setup (ms)	Total (ms)	Per-epoch (ms)	Setup/T (ms)
1	2718	4583	1865	2718
5	2718	10663	1589	544
10	2718	17228	1451	272
50	2718	72618	1398	54

n-scaling. Table 6 reports the Phase 1 and Phases 3–5 wall-times, each from an independent run, at $T = 10$ and $\Delta = 55$. Refresh tracks the deployment capacity $N_{\max}$ rather than the active set n, since $m_\alpha = \lceil (1+\epsilon) \cdot N_{\max}^\alpha \rceil$ is fixed at Setup. A tight deployment should therefore set $N_{\max}$ to the active high-water mark and the roughly 2000-epoch buffer of Section 3.3.

Table 6: n-scaling at $\Delta = 55$ and $T = 10$.

n	Setup (s)	Wall/ep (ms)	Phase 1 (ms)	Phases 3–5 (ms)	Comm./ep (MB)
2^{14}	0.13	442	19	324	1.3
2^{16}	0.20	479	60	323	3.0
2^{18}	0.62	684	251	332	9.6
2^{20}	2.77	1508	1079	332	36.0

B Necessity of adaptive-input security

A non-adaptive proof gives no security statement against an adversary who chooses epoch-$(t+1)$ inputs as a function of the epoch-t transcript. Three attack patterns (i)–(iii) show the gap between such a proof and the threat model an updatable PSU deployment faces, and a structural obstacle (iv) explains why direct lifting also fails.

(i) Encoding-trace attacks. The deletion-trace distinguisher of Theorem 1 is one such pattern, where an adaptive receiver probes the retained encoding trace through Y_{t+1}. A non-adaptive proof that fixes Y_{t+1} in advance says nothing about it.

(ii) OPRF replay-and-correlate. After recovering $X_t \setminus Y_t$ from the epoch-t union, the receiver inserts those elements into Y_{t+1} and forces the sender to re-evaluate the OPRF on receiver-controlled inputs. Any cross-epoch leakage in the OPRF re-randomization then becomes exploitable. Our hybrid model rules this out by abstracting the OPRF as $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$, whereas a non-adaptive proof would not even constrain the distribution over Y_{t+1} on which such leakage could be tested.

(iii) Membership inference by input steering. A receiver suspecting $x^* \in X_{t+1}$ steers Y_{t+1} so that the leaked cardinality $|Y_{t+1} \cup X_{t+1}|$ changes by one exactly when $x^* \in X_{t+1}$. This pattern is almost trivial, yet it shows the broader point. The adaptive choice of Y_{t+1} as a function of prior unions is what the deployment grants, and adaptive-input security is the smallest model that captures it.

(iv) State reuse requires inclusion in the functionality. The remaining gap is structural. UC composition handles environments that derive each input from prior outputs, the standard adaptive-input regime. The obstacle to lifting a single-shot PSU proof is not input adaptivity but the shared state, which must be captured in the ideal functionality before composition applies. Theorem 5 carries out the lifting with that state in scope.

Adaptive-input security should therefore be part of the target model for the state-reusing deployments of Section 1, and single-shot security alone does not justify them.

C Schedule-invariance necessity for verbatim full-encoding PSU

This appendix records a formal counterpart to the deletion-trace attack of Section 3.2. In a *verbatim full-encoding PSU*, the receiver re-encodes its full set into L_t^R each epoch and sends it to the sender rather than an incremental update. A corrupted sender therefore observes the entire epoch-t encoding, and the per-epoch protocol of Section 4.4 is one such instance. The statement below shows that, for this class, adaptive-input security forces L_t^R to depend on the receiver's history only through the active map I_t and the sender-side leakage, with the deletion-trace attack as one concrete witness.

Proposition 2 (Schedule-invariance). *Let Π be a verbatim full-encoding multi-epoch PSU that securely realizes $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$ of Figure 4 against an adaptive-*

input environment. Then the distribution of the epoch-t encoding L_t^R depends on the receiver's update history only through the resulting active map I_t and the sender-side leakage transcript permitted by $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$. That is, any two receiver-side schedules σ^A and σ^B that end at the same active map I_t and induce the same sender-side ideal leakage transcript yield identically distributed L_t^R in the corrupted-sender view, up to $\mathsf{negl}(\lambda)$ statistical distance.

Proof. We argue by contraposition. Suppose two schedules σ^A and σ^B that end at the same active map I_t and induce the same sender-side ideal leakage transcript yield encodings $L_t^R(\sigma^A)$ and $L_t^R(\sigma^B)$ distinguishable with advantage α.

Reduction. The environment $\mathcal{Z}$ samples $b \leftarrow \{A, B\}$ and feeds the receiver σ^b with $X_t = \emptyset$. The sender adversary $\mathcal{A}_S$ runs the distinguisher $\mathcal{D}$ on the extracted encoding $L_t^R(\sigma^b)$. In the real execution, $\mathcal{Z}$ accepts with probability at least $\frac{1}{2}+\frac{\alpha}{2}$. In the ideal execution, the simulator $\mathcal{S}_S$'s view depends only on the sender-side ideal leakage transcript, identical for both values of b by hypothesis, so $\mathcal{D}$ guesses b with probability at most $\frac{1}{2} + \mathsf{negl}(\lambda)$. This gap contradicts adaptive-input security.

Concrete witness. Such a same-leakage pair exists and instantiates the deletion-trace attack. Pick dummies $z_1, z_2 \notin I_t$, enumerate $I_t = \{y_1, \ldots, y_{|I_t|}\}$, and set

$$\sigma^A : \Delta_1^R = \{(\mathsf{add}, y_i)\}_{i=1}^{|I_t|} \cup \{(\mathsf{add}, z_1)\}, \qquad \Delta_2^R = \{(\mathsf{del}, z_1)\},$$
$$\sigma^B : \Delta_1^R = \{(\mathsf{add}, y_i)\}_{i=1}^{|I_t|} \cup \{(\mathsf{add}, z_2)\}, \qquad \Delta_2^R = \{(\mathsf{del}, z_2)\}.$$

Both end at I_t with identical sender-side ideal leakage transcripts—the same $|Y_1|$ and $|Y_2|$, the same receiver-side update cardinalities, and $X_1 = X_2 = \emptyset$ throughout—and differ only in the dummy identity, which $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$ hides from the sender.

Corollary 3. *Call a receiver-side schedule* canonical *for I_t if it reaches I_t directly from the clean initial state, with no intervening deletions or re-insertions. If a canonical schedule $\sigma^\star$ induces the same sender-side ideal leakage transcript as a target schedule, then the target schedule's encoding L_t^R is within $\mathsf{negl}(\lambda)$ statistical distance of the distribution induced by $\sigma^\star$. In particular, if $\sigma^\star$ emits a fresh static $\mathsf{Encode}(\mathsf{pp}, I_t)$, then L_t^R is $\mathsf{negl}(\lambda)$-close to $\mathsf{Encode}(\mathsf{pp}, I_t)$, though not necessarily equal in law.*

Proof. Immediate from Proposition 2, since a canonical schedule sharing the target's sender-side leakage transcript yields an L_t^R within $\mathsf{negl}(\lambda)$ of the target's, and a fresh $\mathsf{Encode}(\mathsf{pp}, I_t)$ whenever $\sigma^\star$ emits one.

D Asymmetric protected set: counterexamples

The receiver queries against $P_t^R = Y_t \cup \Delta_t^{R-}$, which includes its just-deleted keys, while the sender encodes only X_t. The two counterexamples below show why the receiver must include Δ_t^{R-} and the sender must not.

Receiver-side collision. A Phase 5 query against Y_t alone leaks the sender's update identity when both parties delete the same key. Consider $X_0 = \{a\}$ and $Y_0 = \{a, c\}$ with the receiver deleting a: world A has the sender delete a, while world B starts from $X_0 = \{c\}$ and deletes c. The two worlds have identical receiver-side ideal leakage, yet a Phase 5 query against $Y_t = \{c\}$ would reveal a in world A and 0^κ in world B. Querying against $P_t^R = Y_t \cup \Delta_t^{R-}$ instead yields 0^κ in both.

Sender-side mismatch. Instead, suppose that the sender encodes $X_t \cup \Delta_t^{S-}$. Now consider a key a that both parties just deleted, so $a \in \Delta_t^{R-} \cap \Delta_t^{S-}$. Phase 3 reads $m_a = 1$ from the sender's encoding and leaves $a \in U_t$, while a Phase 5 query against P_t^R, which contains a, delivers a dummy rather than removing a. The resulting output therefore contains a, contradicting the ideal output that excludes any jointly deleted key. The sender must encode only X_t.

E Proof of Theorem 4: protocol correctness

We proceed by induction on $t \geq 0$ with the hypothesis $U_t = X_t \cup Y_t$.

Base case ($t = 0$). The one-time base-PSU run of Section 4.3 initializes U_0 as Y_0 and appends each $x \in X_0$ revealed when $x \notin Y_0$, giving $U_0 = X_0 \cup Y_0$ except with probability at most $|X_0| \cdot \varepsilon_{\mathrm{rd}}$ under the per-query random-decoding bound (Assumption 3).

Inductive step ($(t-1) \to t$). Assume $U_{t-1} = X_{t-1} \cup Y_{t-1}$. The receiver updates U_t to U_{t-1}, and we analyze each phase's effect on U_t.

Phase 2 (Δ_t^{R+}). The post-update Y_t contains every $y \in \Delta_t^{R+}$, so each lies in $X_t \cup Y_t$ and adding Δ_t^{R+} unconditionally to U_t is correct.

Phase 3 (Δ_t^{R-}). For $y \in \Delta_t^{R-}$, compatibility gives $y \in Y_{t-1}$ and $y \notin Y_t$. The parties compute $m_y = \mathbf{1}[y \in X_t]$ against L_t^S, which encodes only the sender-active X_t. Under distributional erasure and the per-query random-decoding bound, $\mathsf{Decode}(L_t^S, y) = F_{k^S}(t, y)$ if and only if $y \in X_t$, except with probability at most $\varepsilon_{\mathrm{rd}}$. Since $y \notin Y_t$, we have $m_y = 1$ if and only if $y \in X_t \cup Y_t$. The receiver keeps y in U_t exactly when $m_y = 1$, matching the ideal condition.

Phase 4 (Δ_t^{S+}). For $x \in \Delta_t^{S+}$, compatibility gives $x \notin X_{t-1}$ and $x \in X_t$, so x belongs in $X_t \cup Y_t$. The query returns $z_x = \mathbf{1}[x \in P_t^R]$ with $P_t^R = Y_t \cup \Delta_t^{R-}$. If $z_x = 0$, the receiver appends x, which is correct because $x \notin U_{t-1}$. Compatibility gives $x \notin X_{t-1}$ and $x \notin Y_{t-1}$, since $x \notin P_t^R$ excludes both $x \in Y_t$ and $x \in \Delta_t^{R-}$. If $z_x = 1$, the no-op is correct because x already lies in U_t. When $x \in Y_t$, the cache carry keeps it or Phase 2 added it. When $x \in \Delta_t^{R-}$, Phase 3 kept it with $m_x = \mathbf{1}[x \in X_t] = 1$.

Phase 5 (Δ_t^{S-}). For $x \in \Delta_t^{S-}$, compatibility gives $x \in X_{t-1}$ and $x \notin X_t$, so x belongs in $X_t \cup Y_t$ exactly when $x \in Y_t$. The query returns $z_x = \mathbf{1}[x \in P_t^R]$. If $z_x = 0$, the receiver removes x, which is correct because $x \notin P_t^R$ gives $x \notin Y_t$, so $x \notin X_t \cup Y_t$. If $z_x = 1$, the no-op leaves U_t in the correct state. When $x \in Y_t$, the cache carry keeps it or Phase 2 added it, matching $x \in X_t \cup Y_t$. When $x \in \Delta_t^{R-}$, Phase 3 removed it with $m_x = 0$, matching $x \notin X_t \cup Y_t$.

Cumulative error. Each epoch issues at most $\Delta_{\max}$ base-PSU queries at random-decoding slack $\varepsilon_{\mathrm{rd}}$ each, so a union bound over T epochs yields $T \cdot \Delta_{\max} \cdot \varepsilon_{\mathrm{rd}}$. An additional $\mathsf{negl}(\lambda)$ absorbs $\mathcal{F}_{\mathsf{uOKVS}}$ rank-failure aborts and $\mathsf{Encode}/\mathsf{Decode}$ correctness slack. □

F Proof of Proposition 1: base PSU simulation

For each of the four cases $(\delta, C) \in \{\mathsf{su}, \mathsf{ru}\} \times \{S, R\}$, we exhibit the simulator's transcript and bound its distance from the real view in the $(\mathcal{F}_{\mathsf{so\text{-}OPRF}}, \mathcal{F}_{\mathsf{OT}}^{\mathrm{shared}}, \mathcal{F}_{\mathsf{ssPEQT}})$-hybrid model.

(1) Sender-update, sender-corrupted. The real view of $\mathcal{A}_S$ on a query $x \in \Delta_t^{S+} \cup \Delta_t^{S-}$ consists of (a) the $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ share $x^1 = F_k(t, x) \oplus x^0$ with $x^0 \leftarrow V$ uniform, so x^1 is uniform to $\mathcal{A}_S$; (b) the local decoding $\tilde{v} = \mathsf{Decode}(L_t^R, x)$; (c) the $\mathcal{F}_{\mathsf{ssPEQT}}$ sender output share b^S, uniform in $\{0,1\}$ by spec with $b^S \oplus b^R = z_x$; and (d) the $\mathcal{F}_{\mathsf{OT}}^{\mathrm{shared}}$ sender view, namely the sender's own payload pair $((1, x), (0, 0^\kappa)) \in V_{\mathrm{tag}}^2$ placed in the slot determined by b^S. Since b^S is uniform and independent of z_x as an additive share from $\mathcal{F}_{\mathsf{ssPEQT}}$, the slot containing $(1, x)$ leaks no information about z_x.

$\mathcal{S}_S^{\mathsf{su}}$ samples (a), (c), and (d) as fresh uniforms and the sender's own payload pair, matching the real view, and computes (b) as $\tilde{v} = \mathsf{Decode}(L_t^R, x)$ on the dummy-history L_t^R of its $\mathcal{F}_{\mathsf{uOKVS}}$ copy. The only distinguishing channel is the dummy-history L_t^R. Since the sender-corrupted view depends on P_t^R only through $|Y_t|$ and $|\Delta_t^{R-}|$, which are in the sender's ideal leakage, the dummy L_t^R matches the real one at advantage 0 inside the hybrid. This gives $\mathsf{Adv}_{\mathrm{ideal}}^{\mathrm{base}} = \mathsf{negl}(\lambda)$.

(2) Sender-update, receiver-corrupted. The real view consists of (a) the receiver's $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ share x^0, uniform to $\mathcal{A}_R$ by spec; (b) the $\mathcal{F}_{\mathsf{ssPEQT}}$ receiver output share b^R whose XOR with b^S yields z_x; and (c) the $\mathcal{F}_{\mathsf{OT}}^{\mathrm{shared}}$ receiver output $M_{b^S \oplus b^R} = M_{z_x}$, which is tag bit 1 with payload x when $z_x = 0$ and tag bit 0 with a sentinel when $z_x = 1$.

$\mathcal{S}_R^{\mathsf{su}}$ simulates each sender-update batch as a whole. From the receiver's ideal outputs and local input, it computes the opened payload multisets

$$R_{\mathrm{open}}^{+} = (U_t \setminus U_{t-1}) \setminus \Delta_t^{R+}, \qquad R_{\mathrm{open}}^{-} = (U_{t-1} \setminus U_t) \setminus \Delta_t^{R-}.$$

These are precisely the Phase 4 and Phase 5 sender items with $z_x = 0$, while items hidden by P_t^R appear only as sentinels. Given the leaked $|\Delta_t^{S+}|$ and $|\Delta_t^{S-}|$, it places R_{open}^{+} and R_{open}^{-} uniformly in their phase slots, matching the sender's fresh permutation π, and fills the rest with $z_x = 1$ sentinels. For every slot it invokes the $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ simulator for a uniform x^0 and the $\mathcal{F}_{\mathsf{ssPEQT}}$ simulator on input x^0 with forced output z_x, and the $\mathcal{F}_{\mathsf{OT}}^{\mathrm{shared}}$ receiver view is $(1, x)$ for opened slots and $(0, 0^\kappa)$ otherwise.

The residual slack is the requirement that receiver-simulated keys not accidentally equal honest sender keys, bounded by dummy-collision probability $\Delta_{\max} \cdot |\mathcal{K}|^{-1} = \mathsf{negl}(\lambda)$.

(3) Receiver-update, receiver-corrupted. On $y \in \Delta_t^{R-}$, the real view consists of (a) the sender's uniform $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ share y^0; (b) the receiver's uniform $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ share y^1; (c) the local decoding $\tilde{u} = \mathsf{Decode}(L_t^S, y)$; (d) the $\mathcal{F}_{\mathsf{ssPEQT}}$ receiver share, and (e) the share-open message delivering m_y. By role-swap of (2), $\mathcal{S}_R^{\mathsf{ru}}$ reads $m_y = \mathbf{1}[y \in X_t]$ from the ideal output, simulates y^0, y^1, and the $\mathcal{F}_{\mathsf{ssPEQT}}$ share as uniforms, and delivers the share-open with the correct bit. The dummy-history L_t^S enters only through $\tilde{u}$, pseudorandom for non-stored keys, so it matches at advantage 0 with residual slack $\mathsf{negl}(\lambda)$.

(4) Receiver-update, sender-corrupted. The sender's view contains (a) the share y^0, uniform by $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ spec; (b) the $\mathcal{F}_{\mathsf{ssPEQT}}$ sender share, uniform by spec; and (c) the receiver-side share-open transcript, which the sender does not see. By role-swap of (1), all are reproducible without m_y, so $\mathcal{S}_S^{\mathsf{ru}}$ samples them as uniforms. The honest-side L_t^R is irrelevant, since Phase 3 decodes L_t^S held by the sender, and the advantage is $\mathsf{negl}(\lambda)$.

Uniformity. In all four cases the per-query simulator error inside the ($\mathcal{F}_{\mathsf{so\text{-}OPRF}}$, $\mathcal{F}_{\mathsf{OT}}^{\mathrm{shared}}$, $\mathcal{F}_{\mathsf{ssPEQT}}$)-hybrid is $\mathsf{Adv}_{\mathrm{ideal}}^{\mathrm{base}} = \mathsf{negl}(\lambda)$. The real-world plug-in then adds $\mathsf{Adv}^{\mathsf{ssPEQT}}$ whenever $\mathcal{F}_{\mathsf{ssPEQT}}$ is invoked with forced output, and $\mathsf{Adv}^{\mathsf{OT}}$ only on sender-update queries that invoke $\mathcal{F}_{\mathsf{OT}}^{\mathrm{shared}}$. □

G Proof of Theorem 5: protocol security

We give the full hybrid-argument proof deferred from Section 5. Fix a corruption side $C \in \{S, R\}$. The two cases are symmetric up to role-swap, so we present $C = S$, the corrupted sender, and summarize the receiver case.

Hybrid sequence. Define $H_0, H_1, \ldots, H_T$, where H_t runs the ideal $\mathcal{F}_{\mathsf{UPSU}}^{\text{2-update}}$ for epochs $\le t$ and the real protocol in the ($\mathcal{F}_{\mathsf{uOKVS}}, \mathcal{F}_{\mathsf{so\text{-}OPRF}}, \mathcal{F}_{\mathsf{OT}}^{\mathrm{shared}}, \mathcal{F}_{\mathsf{ssPEQT}}$)-hybrid model for epochs $> t$. H_0 differs from the real world only in how the Setup union $U_0 = X_0 \cup Y_0$ is delivered, and H_T is the ideal execution.

*Base hop (*real $\to H_0$*).* The Setup runs the $|X_0|$ base-PSU queries of Section 4.3. Replacing each query by its Proposition 1 simulator, which for $C = S$ handles the sender-update queries, incurs $|X_0| \cdot \mathsf{Adv}_{\mathrm{ideal}}^{\mathrm{base}}$. Their joint output $U_0 = X_0 \cup Y_0$ matches the ideal functionality's Init output. In addition, Setup produces the honest receiver's epoch-0 encoding L_0^R via $\mathsf{BaseEncode}$ and transmits it to the sender. We treat this transmitted encoding as round $t = 0$ of the stateful adaptive hiding game of Definition 2. The epoch-$1, \ldots, T$ refresh encodings below use rounds $1, \ldots, T$, so the encoding-substitution budget is exactly $\mathsf{Adv}_{T+1}^{\mathcal{F}_{\mathsf{uOKVS}}}$.

Inductive hop ($H_{t-1} \to H_t$). The transition converts epoch t from real-hybrid to ideal through three sub-hops.

Sub-hop (i): encoding refresh. In H_{t-1}, the honest receiver's $\mathcal{F}_{\mathsf{uOKVS}}$ processes the real update. In H_t, the simulator drives it with a size-matched dummy update satisfying the $\mathcal{F}_{\mathsf{uOKVS}}$ compatibility invariants, with dummy keys drawn from an unused subspace of $\mathcal{K}$ so no $\perp$ branch is taken. Because Figure 2 embeds a real PRF F_k, replacing the honest party's real map by the dummy map is computational. Its distinguishing advantage is bounded by the stateful adaptive

hiding game of Definition 2 via Lemma 3. The $T+1$ replacements form a single reduction whose hidden bit governs all encodings, so this sub-hop contributes one $\mathsf{Adv}^{\mathcal{F}_{\mathsf{uOKVS}}}_{T+1}$ charge rather than a per-epoch cost. A cleartext OPRF would instead expose $F_k(t,x)$ and reopen the cross-check ruled out in Remark 1.

Sub-hop (ii): Phase 3 queries. Each $y \in \Delta_t^{R-}$ is replaced by $\mathcal{S}^{\mathsf{ru}}_S$ of Proposition 1. Under this replacement, the sender sees a uniform $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ share y^0 and a uniform $\mathcal{F}_{\mathsf{ssPEQT}}$ share. The sender receives no share-open, since it sends its own $\mathcal{F}_{\mathsf{ssPEQT}}$ share to the receiver, who reconstructs m_y without revealing the final bit. Since neither share carries m_y, the simulator needs no ideal output bit. This costs $|\Delta_t^{R-}| \cdot \mathsf{Adv}^{\mathrm{base}}_{\mathrm{ideal}}$.

Sub-hop (iii): Phase 4–5 queries. Each $x \in \Delta_t^{S+} \cup \Delta_t^{S-}$ is replaced by $\mathcal{S}^{\mathsf{su}}_S$. Under this replacement, the sender sees a uniform $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ share x^1, a uniform $\mathcal{F}_{\mathsf{ssPEQT}}$ share, and the $\mathcal{F}^{\mathrm{shared}}_{\mathsf{OT}}$ sender view. This view is the sender's own payload pair, with the tagged payload in the slot set by the sender's uniform share b^S, and it is independent of z_x because b^S is an additive share from $\mathcal{F}_{\mathsf{ssPEQT}}$. The sender reveals nothing, since the reveal step is receiver-side only. This costs $(|\Delta_t^{S+}| + |\Delta_t^{S-}|) \cdot \mathsf{Adv}^{\mathrm{base}}_{\mathrm{ideal}}$.

Induction. Summing sub-hops (ii) and (iii) over T epochs and the Setup gives $(|X_0| + T \cdot \Delta_{\max}) \cdot \mathsf{Adv}^{\mathrm{base}}_{\mathrm{ideal}}$. Sub-hop (i) is charged once as a single stateful hiding experiment rather than as independent per-epoch reductions, at $\mathsf{Adv}^{\mathcal{F}_{\mathsf{uOKVS}}}_{T+1}$. This single charge covers the epoch-0 encoding from the base hop together with the epoch-$1,\ldots,T$ refresh outputs. The total hybrid-model advantage is therefore

$$(|X_0| + T \cdot \Delta_{\max}) \cdot \mathsf{Adv}^{\mathrm{base}}_{\mathrm{ideal}} + \mathsf{Adv}^{\mathcal{F}_{\mathsf{uOKVS}}}_{T+1} + \mathsf{negl}(\lambda),$$

matching Theorem 5. The $\mathsf{negl}(\lambda)$ absorbs dummy-key collisions and $\mathsf{Encode}/\mathsf{Decode}$ correctness slack, and rank-compatibility of Definition 1 eliminates the $\perp$ branch.

Receiver-corrupted case. The receiver-corrupted case is symmetric with three changes. First, sub-hop (i) simulates the honest sender's $\mathcal{F}_{\mathsf{uOKVS}}$ on a dummy update of size $|\Delta_t^S|$, and the sender's PRF key k^S is unseen by the corrupted receiver. Second, sub-hop (ii) uses $\mathcal{S}^{\mathsf{ru}}_R$, which reads m_y off the ideal output of Proposition 1 and forwards it through the share-open step. Third, sub-hop (iii) uses $\mathcal{S}^{\mathsf{su}}_R$, which reconstructs the opened payloads from the receiver's current state and the ideal union delta. It samples their locations uniformly within the leaked $|\Delta_t^{S+}|$ and $|\Delta_t^{S-}|$ batches and forges the $\mathcal{F}^{\mathrm{shared}}_{\mathsf{OT}}$ receiver view with sentinel outputs in unopened slots. The per-epoch and cumulative bounds are identical.

Real-world bound (Corollary 1). Replacing the corrupted-side $\mathcal{F}_{\mathsf{uOKVS}}$ by its $\mathsf{uBand\text{-}OKVS}$ realization adds one more $\mathsf{Adv}^{\mathcal{F}_{\mathsf{uOKVS}}}_{T+1}$, since the honest side was already charged in sub-hop (i) above, for a total of $2\mathsf{Adv}^{\mathcal{F}_{\mathsf{uOKVS}}}_{T+1}$. Replacing the two $\mathcal{F}_{\mathsf{so\text{-}OPRF}}$ instances, $\mathcal{F}^{\mathrm{shared}}_{\mathsf{OT}}$, and $\mathcal{F}_{\mathsf{ssPEQT}}$ with their realizations further adds $\mathsf{Adv}^{\mathsf{so\text{-}OPRF}}_{T_{\mathsf{su}}} + \mathsf{Adv}^{\mathsf{so\text{-}OPRF}}_{T_{\mathsf{ru}}} + N_{\mathsf{ssPEQT}} \cdot \mathsf{Adv}^{\mathsf{ssPEQT}} + N_{\mathsf{OT}} \cdot \mathsf{Adv}^{\mathsf{OT}}$. These terms combine additively by UC composition [9], which establishes Corollary 1. □