This document provides the full UC security proof that are omitted from the main paper due to space constraints. All materials are provided in a compact but complete form to ensure the soundness and completeness of the UC framework.

Scope and Structure

This document includes: (i) the full ledger functionality $\mathcal{L}_{CMC}$; (ii) the full simulator $\mathcal{S}_{CMC}$; and (iii) auxiliary notes linking the simulator to the proof sketch in the main paper.

Ledger functionality $\mathcal{L}_{CMC}$

The ledger functionality $\mathcal{L}_{CMC}$ interacts with $\mathcal{F}_{CMC}$, $\mathcal{F}_{WT}$

Deposit locking. Upon receiving $(\mathsf{LockWT}, WT, Dep_{WT})$, lock Dep_{WT} as the collateral of WT.

Deposit release. Upon receiving $(\mathsf{ReleaseWT}, pid, WT)$, if pid is marked as successfully settled and no valid evidence against WT has been accepted, release Dep_{WT} to WT.

Evidence submission. Upon receiving $(\mathsf{SubmitEvidence}, pid, WT, \mathsf{etype}, \pi)$ from any party, where $\mathsf{etype} \in \{\mathsf{reverse}, \mathsf{mutual}\}$:

1) If pid is already resolved, ignore the submission.
2) Otherwise, evaluate the abstract predicate $\mathsf{Verify}_{\mathsf{etype}}(pid, WT, \pi)$.

Slashing. If $\mathsf{Verify}_{\mathsf{etype}}(pid, WT, \pi) = 1$, slash Dep_{WT}, transfer it to the reporter, and mark pid as resolved.

Rejection. If $\mathsf{Verify}_{\mathsf{etype}}(pid, WT, \pi) = 0$, ignore the evidence and keep Dep_{WT} locked.

Finalization marker. Upon receiving $(\mathsf{Finalize}, pid)$ from $\mathcal{F}_{CMC}$, mark pid as successfully settled unless it has already been resolved by slashing.

Fig. 1: Ledger functionality $\mathcal{L}_{CMC}$

Simulation $\mathcal{S}_{CMC}$

The simulator $\mathcal{S}_{CMC}$ interacts with the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ while accessing $\mathcal{F}_{CMC}$, $\mathcal{F}_{WT}$, $\{\mathcal{F}_{MC}^{\theta}\}_{\theta \in NUM}$, and $\mathcal{L}_{CMC}$. The ledger is honest, whereas users, supervisors, and the Watchtower WT may be statically corrupted.

Internal state. For each payment identifier pid, the simulator stores a tuple $(NUM, ID, m, t, \mathsf{HoldOK}, \mathsf{UpdOK}, \mathsf{status})$ and a deviation log $\mathsf{Dev}[pid]$. Possible values of status are init, $\mathsf{holding}$, ready, $\mathsf{wt\_done}$, $\mathsf{settled}$, and abort.

Stage I: Transaction type identification.

- *Case I1 (honest payer, honest payer-channel supervisors).* Upon receiving $(\mathsf{pay}, pid, pr, pe, m, t)$, the simulator forwards it to $\mathcal{F}_{CMC}$. If the payment is identified as cross-channel, the simulator initializes the internal record for pid and outputs an acceptance transcript on behalf of the payer-channel supervisor.
- *Case I2 (corrupt payer, honest payer-channel supervisors).* The simulator forwards the request to $\mathcal{F}_{CMC}$ and outputs either acceptance or rejection transcripts according to the ideal decision.
- *Case I3 (honest payer, corrupt payer-channel supervisor).* The simulator forwards the request to $\mathcal{F}_{CMC}$. Any externally inconsistent acceptance or rejection transcript generated by the adversary is recorded as a deviation for potential reverse supervision.
- *Case I4 (corrupt payer and corrupt payer-channel supervisor).* The simulator lets the adversary determine external transcripts while maintaining internal consistency with the ideal functionality.

Stage II: Fund holding and pre-deduction.

- *Case H1 (honest supervisors in all involved channels).* For each channel in NUM, the simulator delivers a holding request on behalf of the payer-channel supervisor. The other supervisor in the channel performs tentative pre-deduction. Upon success, the simulator outputs a holding confirmation and records the channel as successfully held.
- *Case H2 (some channel supervisors corrupt, honest payer).* For honest channels, the simulator behaves as in Case H1. For corrupted channels, the adversary may refuse holding, apply incorrect pre-deduction, or output inconsistent confirmations. All such behavior is recorded in the deviation log. If any honest channel rejects holding, the simulator aborts the payment.
- *Case H3 (corrupt payer, honest supervisors).* If the ideal functionality reaches the holding stage, the simulator proceeds as in Case H1. Concurrent reuse of funds by the payer is prevented by the tentative balance updates in the ideal world, and subsequent requests are simulated as rejected.
- *Case H4 (corrupt payer and corrupt supervisors).* The simulator combines the behaviors of Case H2 and Case H3 and records all detected inconsistencies.

Stage III: Encrypted lookup-table updates.

- *Case U1 (honest supervisors, arbitrary Watchtower).* For each successfully held channel, the simulator constructs a wellformed encrypted update consistent with the ideal state and delivers it to $\mathcal{F}_{WT}$ on behalf of the channel supervisors.

- *Case U2 (corrupt supervisors in some channels, honest Watchtower).* Encrypted updates from honest channels are delivered as in Case U1. Malformed or missing updates from corrupted channels prevent Watchtower from satisfying its execution condition.
- *Case U3 (corrupt Watchtower).* The adversary may drop, reorder, or falsely acknowledge updates. Any discrepancy between submitted updates and Watchtower behavior is recorded as a deviation.

Stage IV: Watchtower assistance.

- *Case W1 (honest Watchtower).* If encrypted updates from all channels are available, the simulator outputs a Watchtower confirmation transcript and marks the payment as completed by Watchtower.
- *Case W2 (corrupt Watchtower, stalling or rejecting).* If the watchtower rejects or stalls despite all conditions being satisfied, the simulator aborts the payment and records the deviation.
- *Case W3 (corrupt Watchtower, premature confirmation).* If Watchtower confirms execution without receiving all required updates, the simulator records a critical inconsistency for later slashing.

Stage V: Local ledger finalization.

- *Case F1 (honest supervisors in all channels).* After Watchtower confirmation, the simulator instructs each involved channel to finalize its local balances and outputs a successful cross-channel payment completion transcript.
- *Case F2 (corrupt supervisors in some channels).* If corrupted supervisors refuse to finalize or output inconsistent local states, the simulator records the deviation and reduces the execution to dispute resolution.

Stage VI: Dispute resolution and slashing.

- *Reverse supervision.* When an inconsistency is detectable by honest users, the simulator constructs abstract evidence and submits it to $\mathcal{L}_{CMC}$ on behalf of the user.
- *Mutual supervision.* When contradictory transcripts from Watchtower or supervisors exist, the simulator submits corresponding evidence on behalf of an honest supervisor.

Fig. 2: Simulation $\mathcal{S}_{CMC}$

Mapping to the Security Proof

This section clarifies how the simulator $\mathcal{S}_{CMC}$ and the ledger functionality $\mathcal{L}_{CMC}$ correspond to the arguments in the security proof presented in Section VI of the main paper.

- **Transaction identification.** The forwarding of payment requests and the distinction between in-channel and cross-channel transactions in the proof sketch correspond to Stage I of $\mathcal{S}_{CMC}$.
- **Fund holding and double-spending prevention.** The tentative fund holding and pre-deduction argument in the proof sketch is realized by Stage II of $\mathcal{S}_{CMC}$, which ensures that concurrent or delayed cross-channel executions cannot reuse the same balance.
- **Watchtower correctness and misbehavior.** The proof arguments regarding correct execution, premature confirmation, rejection, or stalling by the watchtower correspond to Stages III and IV of $\mathcal{S}_{CMC}$.
- **Accountability and slashing.** Any deviation detected in the simulator is reduced to evidence submission and adjudication in $\mathcal{L}_{CMC}$, which matches the accountability claims made in the proof sketch.
- **Successful completion.** The final balance updates discussed in the proof sketch correspond to Stage V of $\mathcal{S}_{CMC}$, where all involved channels finalize their local states after watchtower confirmation.

Overall, each step of the proof sketch has a direct and explicit counterpart in the simulator and ledger specifications provided in this document.

Leakage and Assumptions

This section summarizes the information leakage and modeling assumptions implicitly captured by the ideal functionalities and simulators.

a) Permitted leakage.: The adversary may learn the following information: (i) the public payment identifier pid, (ii) the set of channels involved in a cross-channel payment, (iii) whether a payment is successfully completed or aborted, and (iv) public ledger events, including deposit locking, release, and slashing.

b) Protected information.: Plaintext user identities, channel balances, and intermediate routing information are not revealed. The watchtower observes only encrypted identifiers and encrypted balance updates, which are computationally indistinguishable from random values under the assumed security of the Paillier cryptosystem.

c) Corruption assumptions.: The model assumes static corruption. Users, channel supervisors, and the watchtower may be corrupted, while the ledger functionalities are incorruptible.

d) Network assumptions.: Communication channels are authenticated. The adversary controls message scheduling and delivery but cannot forge messages from honest parties.

Composability Notes

The security definition and proof are given in the Universal Composability framework. By the UC composition theorem, the proposed cross-channel payment protocol remains secure under arbitrary concurrent executions and composition with other UC-secure protocols.

In particular, the protocol can be safely composed with: (i) multiple independent payment channels, (ii) multiple concurrent cross-channel payments, and (iii) other blockchain applications that interact with the same ledger functionalities.

The separation between protocol execution, watchtower assistance, and ledger adjudication ensures that composition does not introduce additional trust assumptions or leakage beyond what is explicitly modeled in the ideal functionalities.