# Public → Private Messaging

This post is a outlines a rough overview of how to implement public → private message passing by modelling it around l1 to l2 message passing.

## Introduction

Right now communication between different function types in unidirectional. We can only make calls to public functions from private ones, and not the other way around.

This post seeks to outline how to transfer information from public execution environments to private ones.

## Background

In this previous document the mental model for interoperability between private and public functions can be briefly summarised as follows:

1. Private functions can read, edit and create private state only.
2. Public functions can read and edit public state, and create private state.
3. Private functions can initiate public function calls, but not the other way around.

We aim to devise a mechanism whereby public calls can create state that can be consumed by private transactions. However, initiating private execution from a public function call will be impossible within the protocol (though external infrastructure could provide automation triggered by events), due to the block builder executing public function calls (not users).

## Proposed Solution

The key to bridging communication between public and private functions is to enable public functions to insert items into the private data tree.

The proposed solution echoes the mechanism for sending l1 to l2 messages. The preimage of these messages is entirely public (calculated on Ethereum), but private functions can consume them and nullify them without any issues.

To maintain privacy, a model similar to l1 to l2 message creation for generating nullifiers can be used:

- When a commitment is created, it is formed with a secretHash.
- To consume the message, the consumer must know the preimage of the secretHash.
- If there is no secret component to nullifier creation, then anyone can create a nullifier for a piece of data without consuming the message, effectively rendering the commitment useless.

## Schema Changes

Public Circuit Public Inputs

- Add new commitments field, similarly to how Private Circuit Public Inputs

## Logic Changes

Circuits

- The public kernel circuit will need to aggregate the newly created commitments, this should not change how the rollup circuits handle the commitments.
- The public kernel circuits will also need to verify the sibling paths for the other calls. (Unless the public VM will be allowed to handle these inclusion proofs. It is not clear whether this will be done in the VM or kernel).

ACVM

- A new oracle call to access a private commitment created within a public execution.
- This is required as it is another note store that is separate to the traditional store (created by trail decrypting notes).

- This will be very similar to the l1 to l2 message consumption oracle calls
- This is required as it is another note store that is separate to the traditional store (created by trail decrypting notes).
- This will be very similar to the l1 to l2 message consumption oracle calls

Archiver

- The archiver will need to store the preimages of these calls.