Unencrypted log issues

Currently the unencrypted logs are vulnerable to “source contract impersonation attack”. That is currently we don’t verify in the kernels that the contract address included in the logs is the address of a contract which actually emitted the logs and this makes it possible for a contract to emit a log on behalf of another one.

Proposed solution

A solution to this problem is simple but it will require us to to refactor how the logs are hashed. We currently hash the logs in app circuit and then we feed only a hash of the logs to the kernel. Then in each kernel iteration we accumulate the hash by computing sha256(previous_logs_hash, current_logs_hash)

(relevant piece of code here). Then the logs get published along with an L2 block on L1 and there the Decoder library is used to recover the final logs hash of all the txs by once again performing the hashing of both the app circuit and the kernel circuits.

With this scheme sequencer is forced to make the logs available on-chain because now it is one of the conditions of block’s validity.

Now that we have the need to access the logs inside kernel circuits to verify the kernel address we will need to move the initial hashing out of the app circuit to the kernel circuit. This will require us to modify PrivateCircuitPublicInputs and treat the logs similarly to commitments and nullifiers.

The following:

struct PrivateCircuitPublicInputs { ... unencrypted_logs_hash: [Field; NUM_FIELDS_PER_SHA256], unencrypted_log_preimages_length: Field, ... }

will become:

struct PrivateCircuitPublicInputs { ... unencrypted_logs: [Field; MAX_NEW_LOGS_PER_CALL], ... }

Once this change is done we can check the address in a log against private_call_public_inputs.call_context.storage_contract_address

.

Encrypted log issues

If we decide to generalize the encrypted logs to contain non-note logs then the issue from above applies to non-note encrypted logs as well. Since the logs are encrypted we can’t easily check the included contract address. One way to address this is to prefix each log with a hash of contract address with randomness. This randomness would then be fed as private input to the kernel circuit and there the private kernel would check that hash(private_call_public_inputs.call_context.storage_contract_address, randomness)

matches the log prefix. Once pixie would obtain the log it would decrypt it and perform the same check. Since pixie needs to get a hold of the randomness the randomness would have to be part of the encrypted log.

We would need to inform a kernel circuit and a pixie whether the log currently being handled is note or not (this verification would not happen for notes). We could tackle this by prefixing the log with a boolean value.

A downside of this solution is that it would significantly increase the size of the encrypted log. On Ethereum logs are very cheap and I think our non-note logs will not get used much unless we use a very cheap data availability solution.

Note that it is desirable to support non-note encrypted logs not only for feature parity between private and public contexts but also because it would allow us to use event macros for notes which would be very elegant (see Rahul’s offsite summary) because it would allow us to robustly generate compute_note_hash_and_nullfiier

function.