PSE-Free PoET

Credits: Mic Bowman

PoET overview

- Proof of: valid enclave, time and wait.
- Wait enforced & attested by the enclave.
- Valid attestation == assurance that wait occurred
  - Incoming block can be forwarded immediately
- Time used as tie-breaker for fork resolution
- Monotonic counters used to match requests to enclave
  - Ensure single outstanding request
  - Ensure correctness over reboots

Clock & Monotonic counters rely on Platform Services. Not available on Xeon class machines.

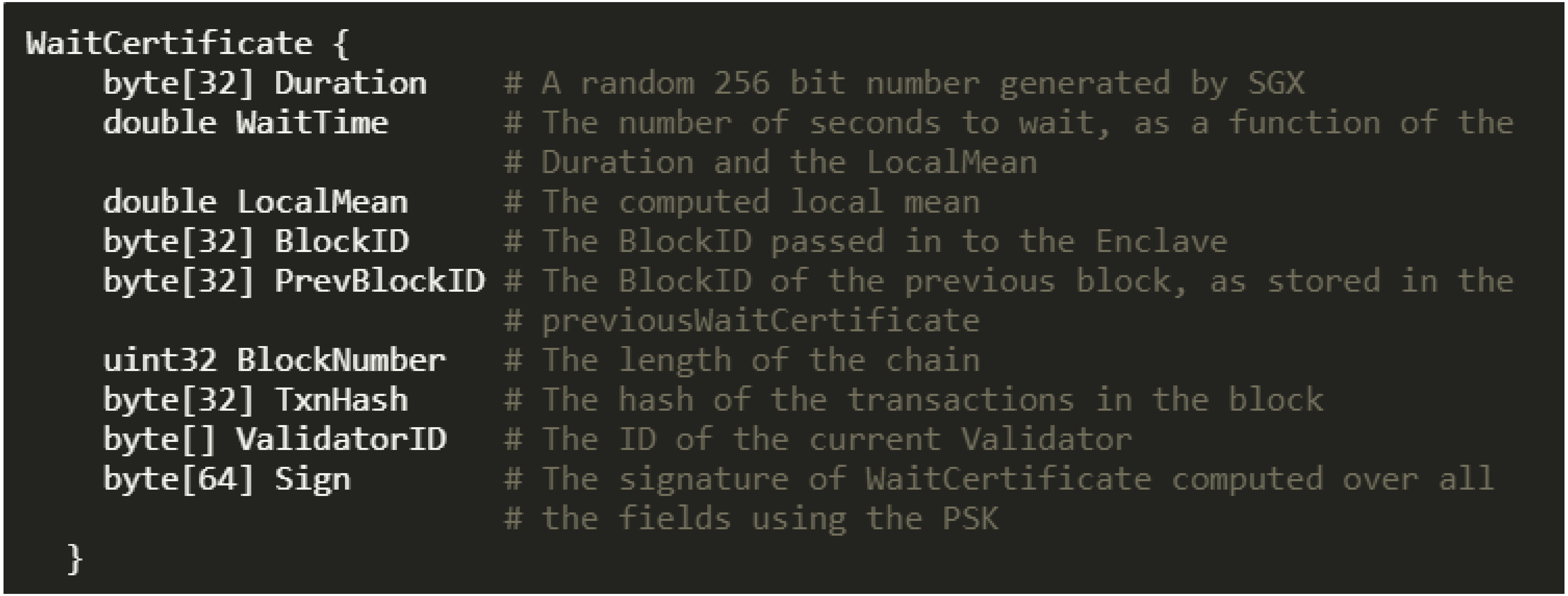
Summary of Changes to PoET

- Enclave creates and registers an ECDSA signing key **every time it is initialized** including **every time the processor is booted**
  - Only one key at a time may be registered for a given EPID pseudonym
  - In addition to the C-test, there is a mandatory delay between registrations for a given EPID pseudonym (like C-test, but R-test throttles registrations rather use)
- Define a “`CreateDuration`” instead of “`CreateWaitTimer`”,
  - Generates a 256 bit random number (uniform distribution), called ‘Duration’
  - Enclave will create at most one number per block number
  - Duration is used to determine time to wait
- “`CreateWaitCertificate`” creates an SGX signed certificate containing the Duration
- The community of validators enforces the wait
  - Wait is determined from block clock/wall clock synchronization
  - All handling of the local mean occurs outside the enclave

Enclave Initialization

- Enclave creates a new ECDSA key pair on initialization
  - There is no option to load an old key pair from sealed storage
- The Enclave’s public key must be registered with the ledger
  - Sign up process is the same as before, at least K blocks must have passed since the last time a key was registered for the server
  - The enclave may not validate a block until at least C blocks have been added to the chain since the enclave was registered.
  - PERFORMANCE IMPLICATION → One IAS request and one registration transactions per boot
- Enclave keeps in memory the ECDSA key pair and a table mapping block number to wait certificates

WaitCertificate composition



```
WaitCertificate {
    byte[32] Duration    # A random 256 bit number generated by SGX
    double WaitTime      # The number of seconds to wait, as a function of the
                         # Duration and the LocalMean
    double LocalMean     # The computed local mean
    byte[32] BlockID     # The BlockID passed in to the Enclave
    byte[32] PrevBlockID # The BlockID of the previous block, as stored in the
                         # previousWaitCertificate
    uint32 BlockNumber   # The length of the chain
    byte[32] TxnHash     # The hash of the transactions in the block
    byte[] ValidatorID   # The ID of the current Validator
    byte[64] Sign        # The signature of WaitCertificate computed over all
                         # the fields using the PSK
  }
```

WallClock & ChainClock

- WallClock (WC)
  - Maintained independently by each validator
  - Number of seconds since the validator's 'synchronization event'
  - Real time, reasonably accurate but shouldn't drift excessively
- ChainClock (CC)
  - Sum of Durations of all blocks since the synchronization event
  - In practice, sum of 'Wait Times' computed from the Duration
- Block Eligible for consensus if CC <= WC

Validator enforced wait

- Upon WaitCertificate creation, originating validator computes WaitTime from Duration & waits
- Block is forwarded after WaitTime seconds
- On neighboring validators:
  - New ChainClock CC’ = CC+WaitTime
  - If CC’ <= WC, block is eligible for consensus
  - Else (early arriving block), wait until WC catches up
- The wait is primarily for the purpose of improving efficiency

Block Publishing

- Start assembling the proposed block
- Get the Duration from the enclave and compute WaitTime
- Set a timer (outside SGX)
- On timer expiration, stop assembling the block, get transaction details
- Compute WaitCertificate with transaction details
- Publish Block, WaitCertificate

Block verification, Leader election

- Incoming block verification:
    - WaitCertificate verification
    - C, Z, K, R tests
    - CC’ <= WC
- For eligible blocks, a winning block should:
    - Extend the longest valid fork
    - Fork resolution order:
        - Longest valid fork
        - Shortest CC’
        - Tie breaker: smallest Duration value

Bootstrapping new validators

- What happens if a new validator joins an existing network?
- When do we start WC & CC?
- How can the validator catch up?

- On registration, define a ‘Synchronization block’: Random block between validator registration & Cth block
- Start building chain by requesting blocks from neighbors
- When Synchronization Block is received, WC = CC = 0
- No Eligibility checks before Cth block (all prior blocks are ‘Eligible’)
- At the end of C block delay:
  - Start creating new block
  - Start Validator Enforced Wait (i.e. enforce CC’ <= WC)