

# Native Randomness on Sui

George Digkas, Mysten Labs



# Many applications depend on randomness

- **NFTs**

Randomly assigning attributes when minting NFTs, such as rarity levels or unique features.

- **Gaming**

RNG elements in battles, loot boxes, generation of game environments, crypto-kitties fair gene mutation.

- **Gambling**

Decentralized lotteries, casinos and card games → ensure fair play.

- **Protocols**

Electing rotating leaders or distributing stake yield randomly to participants in a network.

- **Marketing Campaigns**

Running lucky draws or fan rewards programs where winners are chosen randomly.

- **Security Processes**

Randomly assigning duties or resources: court judges, auditors to firms, IRS random sampling, verifiable LLMs (seed).



# Randomness

Historically, *high-quality randomness* required either:

- Local generation  
(resource-intensive and potentially costly)
- Reliance on an external trusted source  
(inherently risky and challenging)

Our Goal: Develop a decentralized system for randomness that is both unpredictable and unbiased, eliminating reliance on costly local generation or risky external sources.

# Existing solutions

Various approaches, with different tradeoffs:

- commit-and-reveal (biasable)
- VDFs (slow)
- VRFs (requires 100% liveness + biasable)

# Native randomness on Sui

- Decentralized randomness
- Simple, one-step API
- No trust external oracles
- Secure
- Lightning fast

# API

# Sui Move Functions: Random Value Generation

- **`generate_u8(g: &mut RandomGenerator): u8`**
- **`generate_u16(g: &mut RandomGenerator): u16`**
- **`generate_u32(g: &mut RandomGenerator): u32`**
- **`generate_u64(g: &mut RandomGenerator): u64`**
- **`generate_u128(g: &mut RandomGenerator): u128`**
- **`generate_u256(g: &mut RandomGenerator): u256`**

# Sui Move Functions: Range-Based Random Value Generation

- `generate_u8_in_range(g: &mut RandomGenerator, min: u8, max: u8 ) : u8`
- `generate_u16_in_range(g: &mut RandomGenerator, min: u16, max: u16 ) : u16`
- `generate_u32_in_range(g: &mut RandomGenerator, min: u32, max: u32 ) : u32`
- `generate_u64_in_range(g: &mut RandomGenerator, min: u64, max: u64 ) : u64`
- `generate_u128_in_range(g: &mut RandomGenerator, min: u128, max: u128 ) : u128`



# Sui Move Functions

## Random Value Generation (bytes)

- **generate\_bytes** (g: &mut RandomGenerator, num\_of\_bytes: u16) :  
vector<u8>

## Boolean Value Generation

- **generate\_bool** (g: &mut RandomGenerator) : bool

# Developer Guide

# Access on-chain randomness

## Simple example: roll dice

```
entry fun roll_dice(r: &Random, ctx: &mut TxContext): u8
{
    let generator = new_generator(r, ctx);
    random::generate_u8_in_range(
        &mut generator, 1, 6
    )
}
```

Define function as entry

# Use (non-public) entry functions

```
public fun play_dice(  
    guess: u8,  
    fee: Coin<SUI>,  
    r: &Random,  
    ctx: &mut TxContext  
): Option<GuessedCorrectly> {  
    assert!(coin::value(&fee) == 1000000, EInvalidAmount);  
    transfer::public_transfer(fee, CREATOR_ADDRESS);  
    let generator = new_generator(r, ctx);  
    if (guess == random::generate_u8_in_range(&mut generator, 1, 6)) {  
        option::some(GuessedCorrectly {})  
    } else {  
        option::none()  
    }  
}
```

# An attacker can deploy the next function:

```
public fun attack(  
    guess: u8,  
    fee: Coin<SUI>,  
    r: &Random,  
    ctx: &mut TxContext  
): GuessedCorrectly {  
    let output = dice::play_dice(guess, r, fee, ctx);  
    // will revert the transaction if roll_dice returned option::none()  
    option::extract(output)  
}
```

# Use (non-public) entry functions

```
entry fun play_dice(  
    guess: u8,  
    fee: Coin<SUI>,  
    r: &Random,  
    ctx: &mut TxContext  
) : Option<GuessedCorrectly> {  
    assert!(coin::value(&fee) == 1000000, EInvalidAmount);  
    transfer::public_transfer(fee, CREATOR_ADDRESS);  
    let generator = new_generator(r, ctx);  
    if (guess == random::generate_u8_in_range(&mut generator, 1, 6)) {  
        option::some(GuessedCorrectly {})  
    } else {  
        option::none()  
    }  
}
```

# PTBs restrictions

```
public fun attack(output: Option<GuessedCorrectly>): GuessedCorrectly {  
    option::extract(output)  
}
```

Automatically enforce PTB restrictions to prevent composition attacks at the PTB level.

SUI rejects PTBs that have commands that are not **TransferObjects** or **MergeCoins** following a **MoveCall** command that uses **Random** as an input.



Generate randomness using  
function-local RandomGenerator

# Instantiating RandomGenerator

`RandomGenerator` is secure as long as it was created by the consuming module. In case it is passed as an argument, the caller might be able to predict the outputs of that `RandomGenerator` instance (e.g., by calling `bcs::to_bytes(&generator)` and parsing its internal state).

Move compiler prevents defining public functions with `RandomGenerator` as an argument.

Make sure that the “unhappy path” of your function does not charge more gas than the “happy path”

# Resource Equivalence Principle for Preventing Limits-Based Attacks

```
entry fun play(r: &Random, payment: Coin<SUI>, ...) {  
    ...  
    let win = random::generate_u8_in_range(r) & 2;  
    if (win == 1) { // happy flow  
        ... cheap computation ...  
    } else {  
        ... expensive computation ...  
    }  
}
```

# Resource Equivalence Principle for Preventing Limits-Based Attacks

To prevent **limits-based attacks**, your randomness-using function must not use more resources in the "**unhappy**" path than the "**happy**" path.

# Recap

- Define your function as (private) **entry**.
- Prefer generating randomness using function-local **RandomGenerator**.
- Make sure that the "**unhappy path**" of your function does not charge more gas than the "**happy path**".

# Demo



# Randomness Examples

- Workshop code repo:

<https://github.com/MystenLabs/sui-native-randomness>

- More Examples:

[https://github.com/MystenLabs/sui/tree/main/sui\\_programmability/examples/games/sources](https://github.com/MystenLabs/sui/tree/main/sui_programmability/examples/games/sources)



# Special Thanks

- Andrew Schran
- Ben Riva

# Thank you

