

One. Thousand

The world's first decentralized lottery.

From the makers of **One. Ether**. Release Date: September 30, 2015

Why?

We wanted to highlight the strengths of Ethereum by showing the world what could be done that is unique to Bitcoin.

Remember what happened to [BitLotto?](#) Or the huge array of Bitcoin exit scams, and for that matter, a number of Ethereum scams that used centralized protocols? This is no more.

We're bringing to the table a decentralized lottery highlighting the power of Ethereum that is:

- **Publicly auditable:** no tricks up our sleeve here. The source code is on the blockchain, in solidity.
- **Provably fair:** the hashes of the server seed is released publicly. If we don't supply a seed in 24 hours, the system self destructs, and everyone can get their funds back.
- **Easy to use:** no downloads, no accounts, just send one transaction and see your purchase on the site

Now we need you.

Help us show the world via one of Ethereum's first working lottery Dapps how powerful our community is by helping us raise some successful rounds!

<http://oneether.com/thousand>

How is the winner decided?

There are 3 components:

1. A server secret of which the hash is revealed.
2. A client seed that is continually updated for each 1000 tickets.
3. The blockhash of the block following the end of a round.

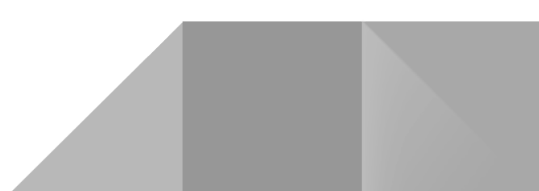
The sha3 of the server secret and client seed, XOR'ed with the blockhash, mod 1000 is the index of the winner.

What happens if we withhold the secret?

The contract specifies that, if within 24 hours of the round end, if we don't supply the secret, all players are returned their money via the `recoverLostFunds()` function.

Why can't players cheat?

Even if you mine, you can't cheat, because part of the random number generator is based on the server secret, to which only we have access.



Why can't we cheat?

We can't cheat, at least meaningfully, because:

1. We would need to be the last transaction in a round consistently, **and**
2. We would need to have enough hashrate to consistently mine the block after the round ends, **and**
3. We would be risking the reputation of One. Ether as a whole.

In short, it is very difficult to forge all three.

Once the round ends, the client seed and blockhash are set in stone, so the time at which we release the server secret has no impact on the final decision.

Appendix 1: Solidity source code

```
contract TheLuckyOne {  
  
    address owner;  
  
    mapping(uint => address) public tickets; //All tickets, ever, in order  
    mapping(uint => uint) public winners; //All winners, ever, in order  
  
    uint public numTickets; //Number of tickets bought lifetime  
    uint public numWinners; //Number of pots awarded lifetime  
    uint public lastProcessed; //Last index of tickets processed
```

```
uint public lastBlock; //Last block of the round to be processed
```

```
bytes32 public clientSeed; //Client seed
```

```
bytes32 public curSecretHash; //Hash of server seed
```

```
bytes32 public lastBlockHash; //Block hash to use
```

```
bytes32 public lastClientSeed; //Client seed to use
```

```
uint public constant TICKETSPERROUND = 1000; //Number of tickets per round
```

```
uint public constant TIMEOUT = 24 hours;
```

```
uint public constant ETHERVAL = 10000000000000000000;
```

```
uint public deadlineStart; //Value of 0 means no deadline
```

```
event Log(uint value);
```

```
function TheLuckyOne(bytes32 initialSecretHash) {
```

```
    curSecretHash = initialSecretHash;
```


```
    owner = msg.sender;

}

//Withdraw excess funds, keeping enough for payouts
function adminWithdraw() {
    if (msg.sender != owner) {
        return;
    }

    //Can't withdraw the amount needed for payout
    uint frozenValue = ((numTickets - lastProcessed) * ETHERVAL);

    //Withdraw the rest
    Log(this.balance - frozenValue);
    msg.sender.send(this.balance - frozenValue);
}
```



//Get time in seconds since deadline, returns 1337 if no deadline

```
function getTimeElapsed() constant returns(uint) {
```

```
    if (deadlineStart == 0) {
```

```
        return 1337;
```

```
    } else {
```

```
        return block.timestamp - deadlineStart;
```

```
    }
```

```
}
```

//Clone of sha3

```
function sha3clone(bytes32 input) constant returns (bytes32) {
```

```
    return sha3(input);
```

```
}
```

//Return funds to players if server seed not released in time

```
function recoverLostFunds() {
```

```
    //No deadline
```

```
    if (deadlineStart == 0) {
```


```
    return;
}

//Deadline not hit yet
if (block.timestamp - deadlineStart < TIMEOUT) {
    return;
}

//Destroy the hash to prevent supplying a working seed at this phase
curSecretHash = 0;

uint toSend = 0;

//Calculate how much to pay based off tickets
for (var i = lastProcessed; i < numTickets; i++) {
    if (tickets[i] == msg.sender) {
        toSend += ETHERVAL;
    }
}
}
```



```
//Refund full ticket value  
msg.sender.send(toSend);  
}
```

```
//Reveal seed and pay out last round
```

```
function revealAndPayout(bytes32 curSecret, bytes32 nextSecretHash) {
```

```
    //Not admin
```

```
    if (msg.sender != owner) {
```

```
        return;
```

```
    }
```


```
    //Invalid Seed
```

```
    if (sha3(curSecret) != curSecretHash || curSecretHash == 0) {
```

```
        return;
```

```
    }
```

```
    //Payout premature
```




```
if (numTickets < TICKETSPERROUND ||
    lastProcessed > numTickets - TICKETSPERROUND) {
    return;
}

//Fetch the non-server sources of entropy
bytes32 serverClientHash = sha3(curSecret, lastClientSeed);

//Calculate winner and pay out
uint winnerIdx =
    lastProcessed + uint(serverClientHash ^ lastBlockHash) % TICKETSPERROUND;
tickets[winnerIdx].send(TICKETSPERROUND * ETHERVAL);
Log (winnerIdx);

//Record win
winners[numWinners++] = winnerIdx;

//Update state variables for next pot
```



```
lastProcessed += TICKETSPERROUND;

//Reset deadline
deadlineStart = 0;


//Update seed
curSecretHash = nextSecretHash;

}

function () {

    uint ticketsBought;

    //Calculate tickets bought
    //Price is 1.025 ETH per ticket for 25+ tickets
    if (msg.value >= 25625 finney) {
        ticketsBought = msg.value/10250000000000000000;
```



```
//Refund difference

msg.sender.send(msg.value % 1025000000000000000);

}

//Price is 1.050 ETH per ticket for 10+ tickets

else if (msg.value >= 10500 finney) {

    ticketsBought = msg.value/10500000000000000000;

    //Refund difference

    msg.sender.send(msg.value % 10500000000000000000);

}

//Price is 1.100 ETH per ticket for 5+ tickets

else if (msg.value >= 5500 finney) {

    ticketsBought = msg.value/11000000000000000000;

    //Refund difference


    msg.sender.send(msg.value % 11000000000000000000);

}

//Price is 1.150 ETH per ticket for 1+ tickets

else {

    ticketsBought = msg.value/11500000000000000000;
```



```
//Refund difference

msg.sender.send(msg.value % 115000000000000000);

}

//Update client seed

clientSeed = sha3(clientSeed, msg.sender, msg.value);

//If this is the final ticket, record the block and set the deadline

if (numTickets/TICKETSPERROUND != (numTickets + ticketsBought)/TICKETSPERROUND)
{

    lastBlock = block.number;

    lastClientSeed = clientSeed;


    deadlineStart = block.timestamp;

}

//Update blockhash (done here because of 256 timeout)

if (block.blockhash(lastBlock + 1) != 0) {

    lastBlockHash = block.blockhash(lastBlock + 1);
```



```
}  
  
//Buys a ticket for each ether  
for (var i = 1; i <= ticketsBought; i += 1) {  
    tickets[numTickets++] = msg.sender;  
}  
  
}  
  
}
```