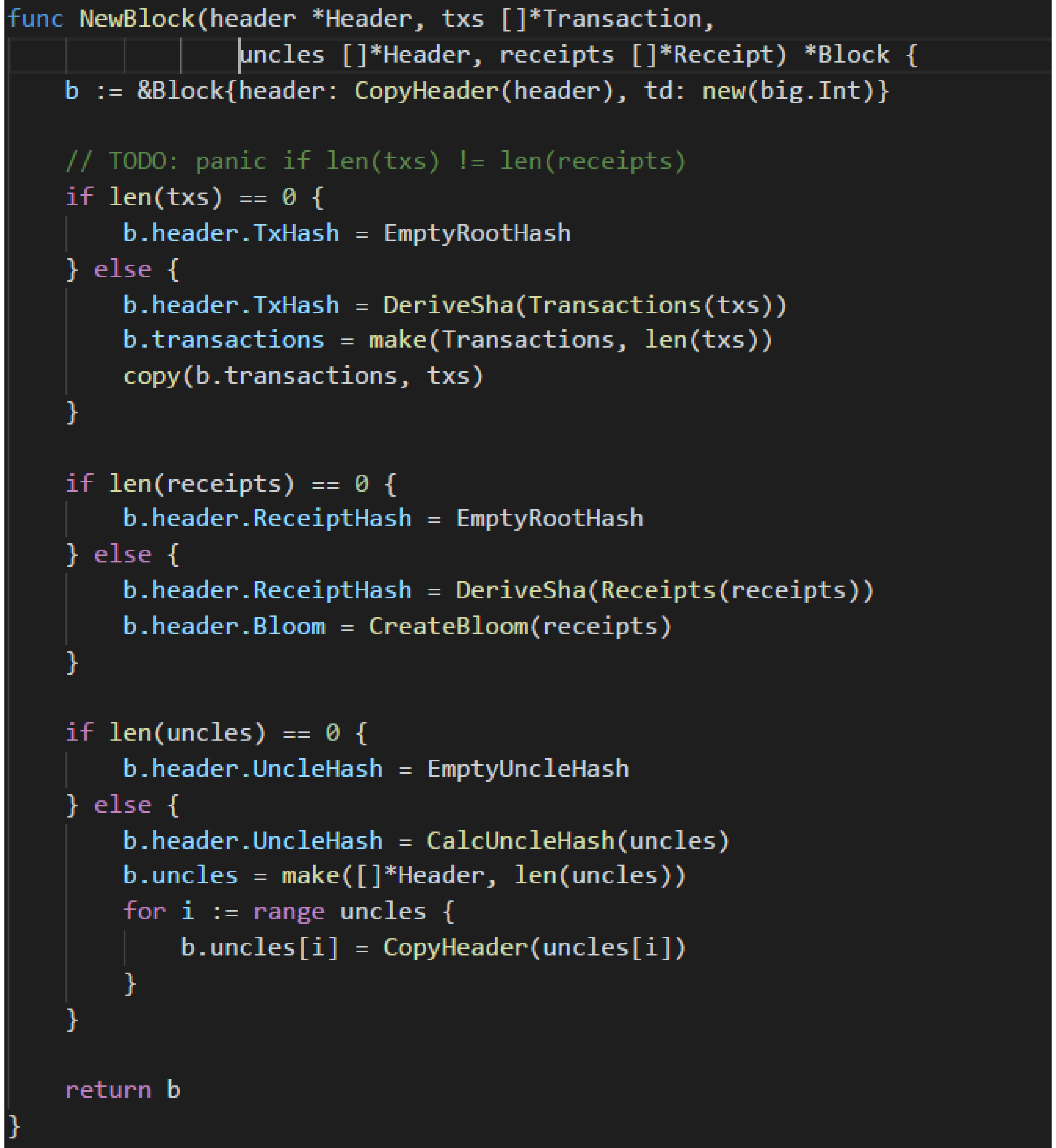
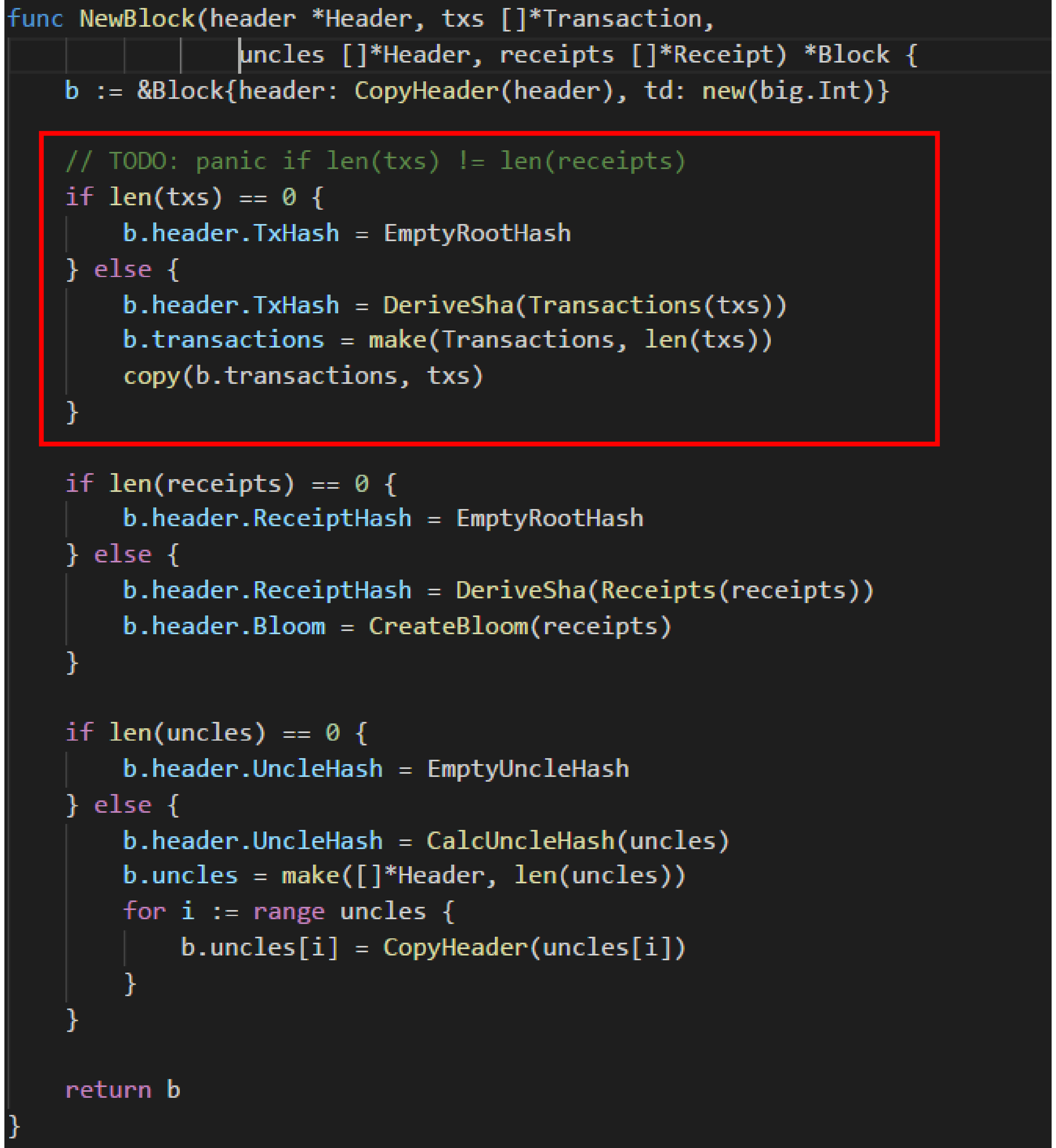


```go
func NewBlock(header *Header, txs []*Transaction,
				uncles []*Header, receipts []*Receipt) *Block {
	b := &Block{header: CopyHeader(header), td: new(big.Int)}

	// TODO: panic if len(txs) != len(receipts)
	if len(txs) == 0 {
		b.header.TxHash = EmptyRootHash
	} else {
		b.header.TxHash = DeriveSha(Transactions(txs))
		b.transactions = make(Transactions, len(txs))
		copy(b.transactions, txs)
	}

	if len(receipts) == 0 {
		b.header.ReceiptHash = EmptyRootHash
	} else {
		b.header.ReceiptHash = DeriveSha(Receipts(receipts))
		b.header.Bloom = CreateBloom(receipts)
	}

	if len(uncles) == 0 {
		b.header.UncleHash = EmptyUncleHash
	} else {
		b.header.UncleHash = CalcUncleHash(uncles)
		b.uncles = make([]*Header, len(uncles))
		for i := range uncles {
			b.uncles[i] = CopyHeader(uncles[i])
		}
	}

	return b
}
```

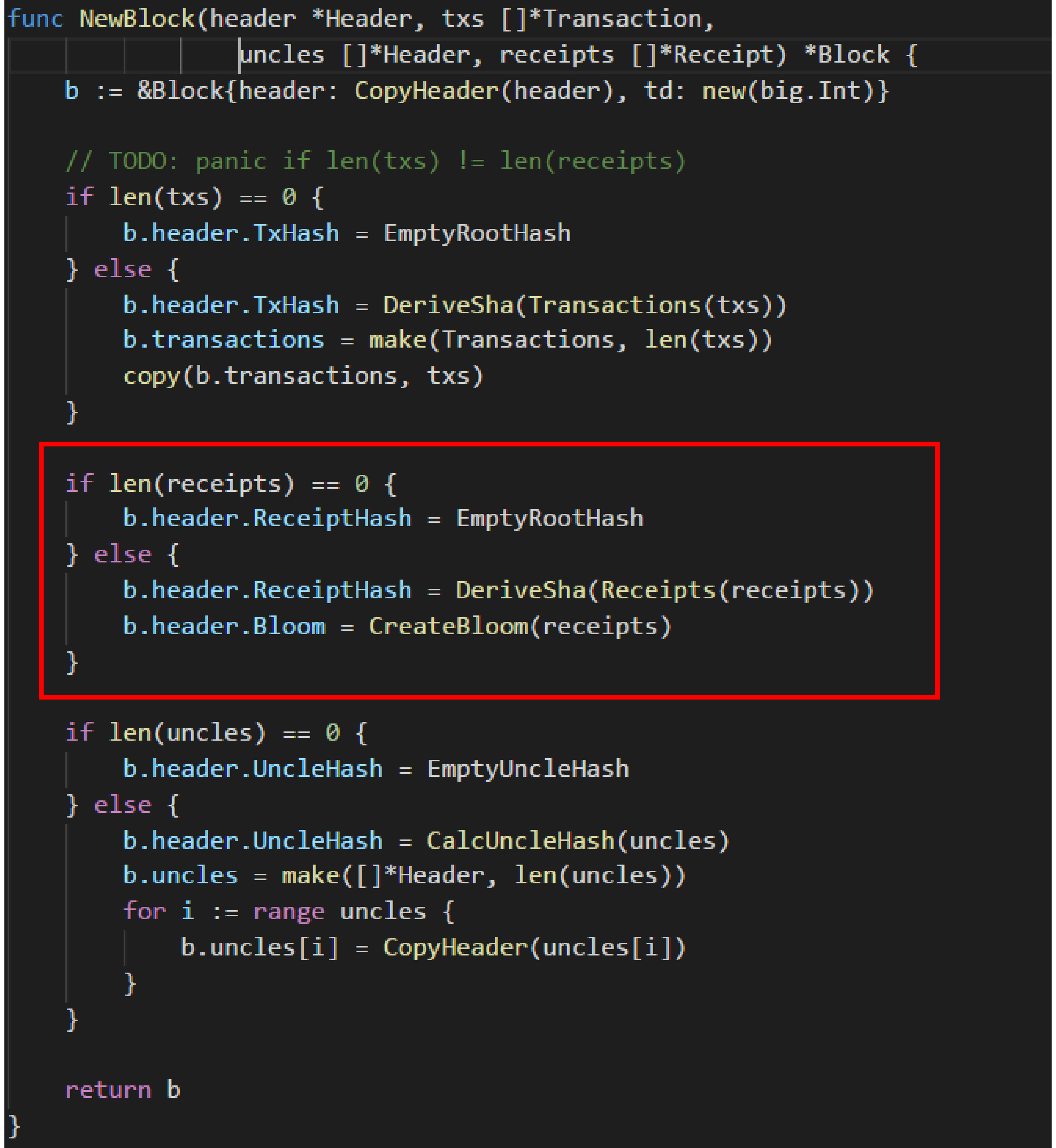
block.go中，NewBlock函数里调用DeriveSha来得到交易树和收据树的根哈希值



```go
func NewBlock(header *Header, txs []*Transaction,
				uncles []*Header, receipts []*Receipt) *Block {
	b := &Block{header: CopyHeader(header), td: new(big.Int)}

	// TODO: panic if len(txs) != len(receipts)
	if len(txs) == 0 {
		b.header.TxHash = EmptyRootHash
	} else {
		b.header.TxHash = DeriveSha(Transactions(txs))
		b.transactions = make(Transactions, len(txs))
		copy(b.transactions, txs)
	}

	if len(receipts) == 0 {
		b.header.ReceiptHash = EmptyRootHash
	} else {
		b.header.ReceiptHash = DeriveSha(Receipts(receipts))
		b.header.Bloom = CreateBloom(receipts)
	}

	if len(uncles) == 0 {
		b.header.UncleHash = EmptyUncleHash
	} else {
		b.header.UncleHash = CalcUncleHash(uncles)
		b.uncles = make([]*Header, len(uncles))
		for i := range uncles {
			b.uncles[i] = CopyHeader(uncles[i])
		}
	}

	return b
}
```

创建交易树，计算根哈希值

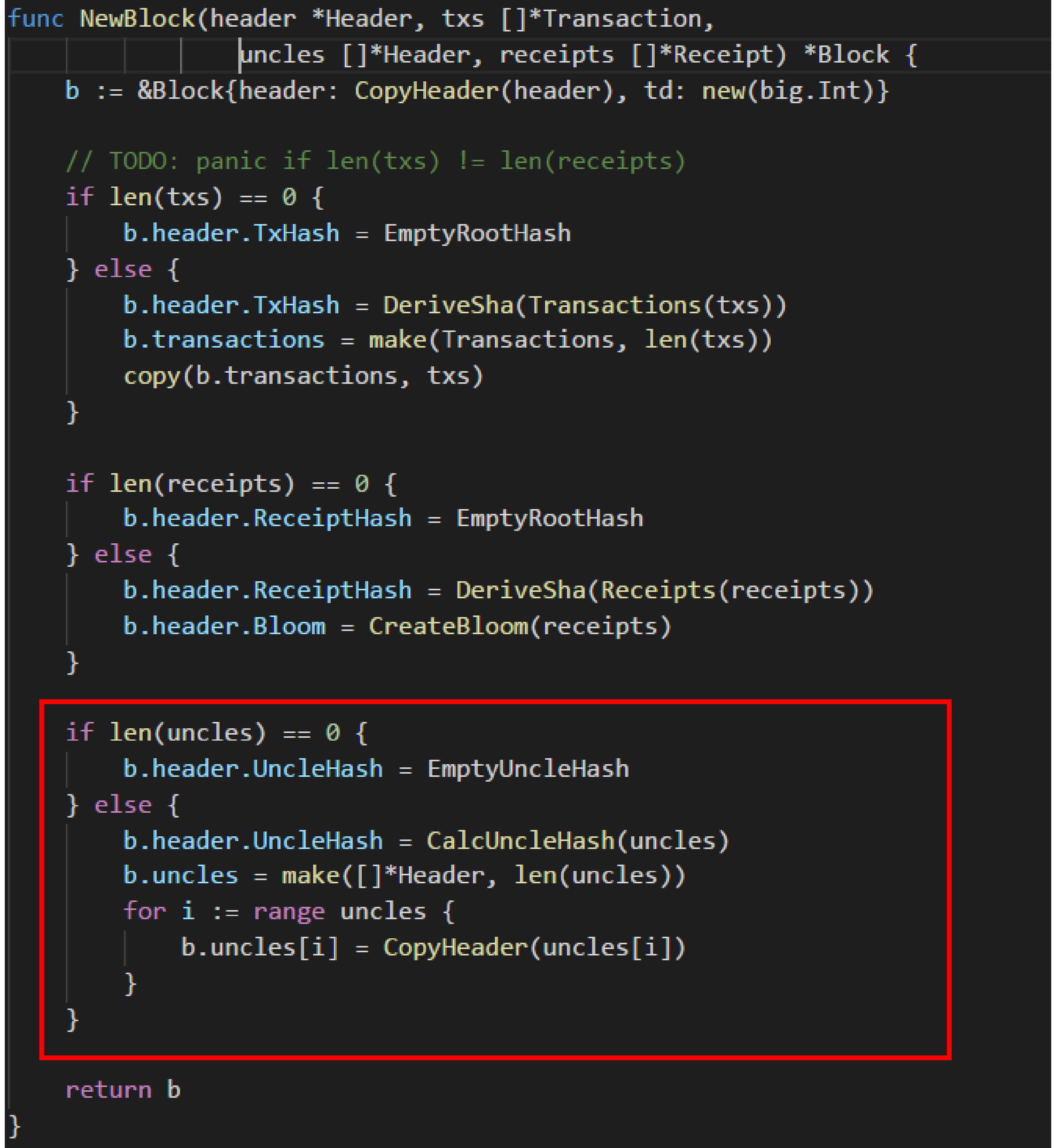
创建交易列表



```go
func NewBlock(header *Header, txs []*Transaction,
                    uncles []*Header, receipts []*Receipt) *Block {
    b := &Block{header: CopyHeader(header), td: new(big.Int)}

    // TODO: panic if len(txs) != len(receipts)
    if len(txs) == 0 {
        b.header.TxHash = EmptyRootHash
    } else {
        b.header.TxHash = DeriveSha(Transactions(txs))
        b.transactions = make(Transactions, len(txs))
        copy(b.transactions, txs)
    }

    if len(receipts) == 0 {
        b.header.ReceiptHash = EmptyRootHash
    } else {
        b.header.ReceiptHash = DeriveSha(Receipts(receipts))
        b.header.Bloom = CreateBloom(receipts)
    }

    if len(uncles) == 0 {
        b.header.UncleHash = EmptyUncleHash
    } else {
        b.header.UncleHash = CalcUncleHash(uncles)
        b.uncles = make([]*Header, len(uncles))
        for i := range uncles {
            b.uncles[i] = CopyHeader(uncles[i])
        }
    }

    return b
}
```

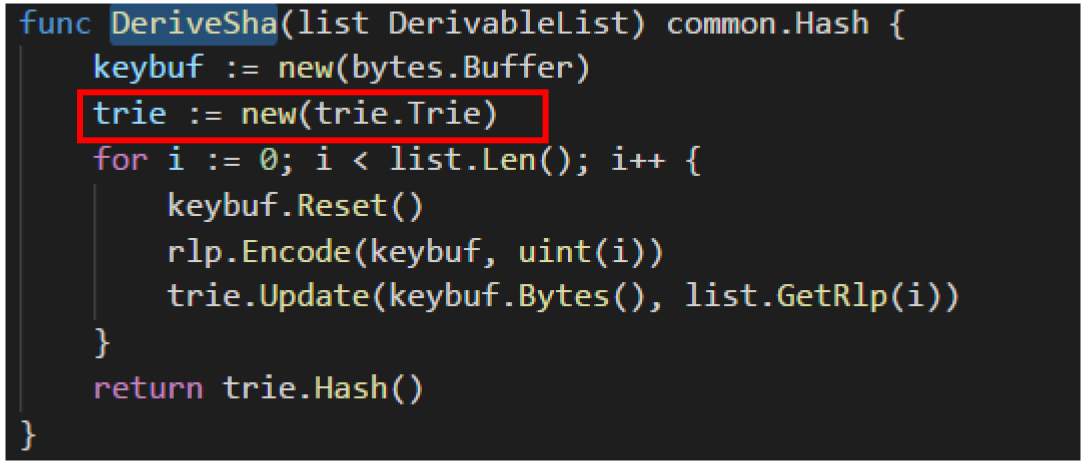
创建收据树，计算根哈希值

创建bloom filter



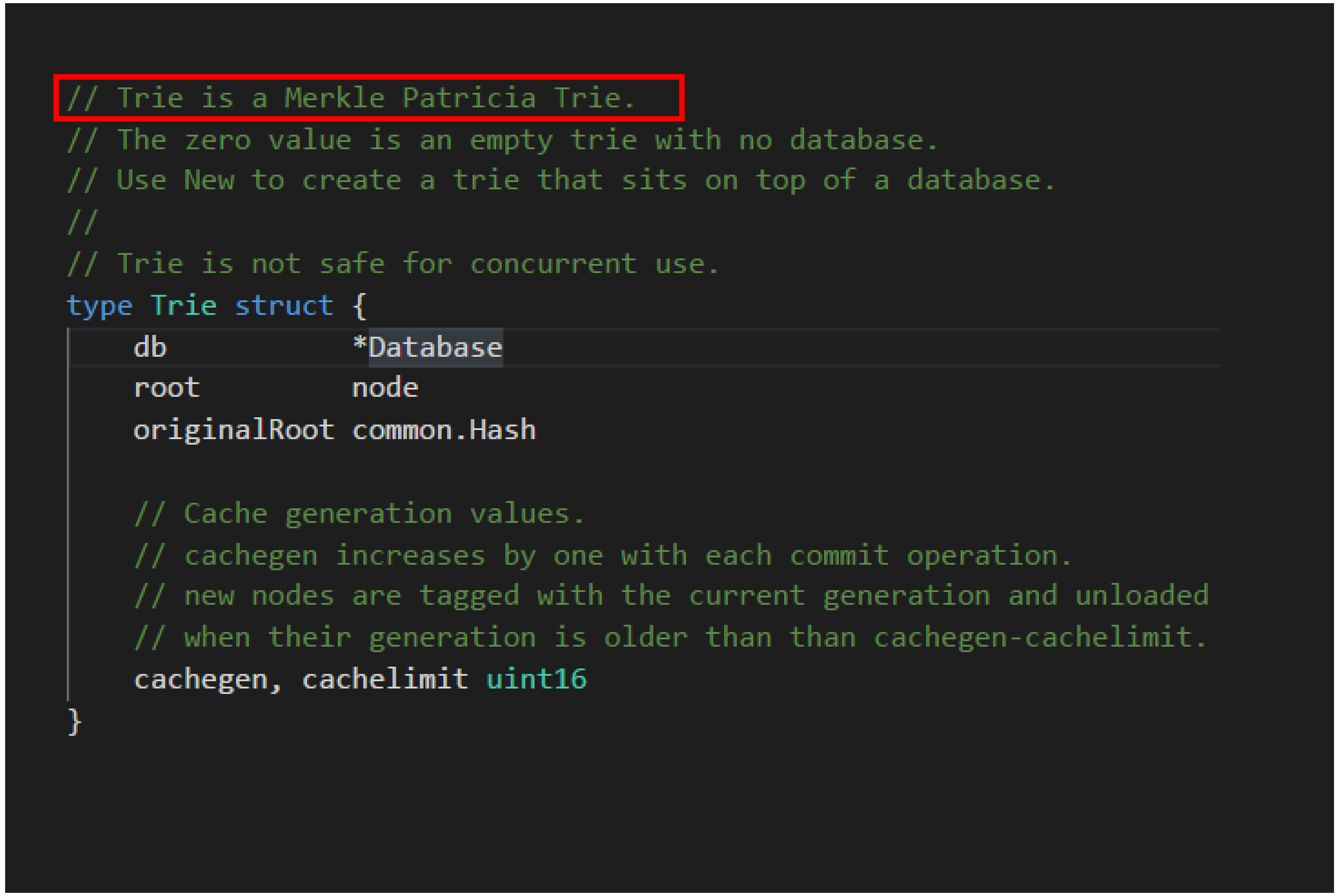
```go
func NewBlock(header *Header, txs []*Transaction,
			uncles []*Header, receipts []*Receipt) *Block {
	b := &Block{header: CopyHeader(header), td: new(big.Int)}

	// TODO: panic if len(txs) != len(receipts)
	if len(txs) == 0 {
		b.header.TxHash = EmptyRootHash
	} else {
		b.header.TxHash = DeriveSha(Transactions(txs))
		b.transactions = make(Transactions, len(txs))
		copy(b.transactions, txs)
	}

	if len(receipts) == 0 {
		b.header.ReceiptHash = EmptyRootHash
	} else {
		b.header.ReceiptHash = DeriveSha(Receipts(receipts))
		b.header.Bloom = CreateBloom(receipts)
	}

	if len(uncles) == 0 {
		b.header.UncleHash = EmptyUncleHash
	} else {
		b.header.UncleHash = CalcUncleHash(uncles)
		b.uncles = make([]*Header, len(uncles))
		for i := range uncles {
			b.uncles[i] = CopyHeader(uncles[i])
		}
	}

	return b
}
```

计算叔父区块的哈希值，构建叔父数组



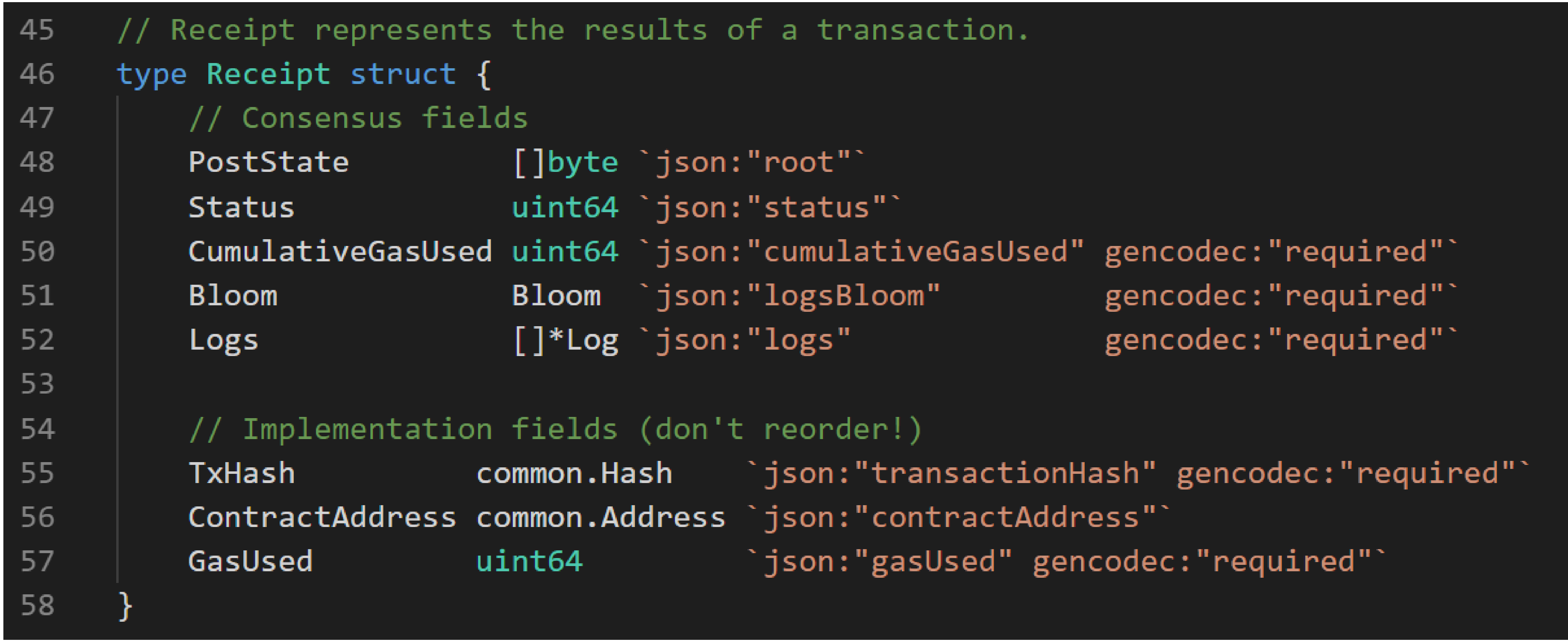
```go
func DeriveSha(list DerivableList) common.Hash {
    keybuf := new(bytes.Buffer)
    trie := new(trie.Trie)
    for i := 0; i < list.Len(); i++ {
        keybuf.Reset()
        rlp.Encode(keybuf, uint(i))
        trie.Update(keybuf.Bytes(), list.GetRlp(i))
    }
    return trie.Hash()
}
```

derive_sha.go中，DeriveSha函数把Transactions和Receipts建为trie



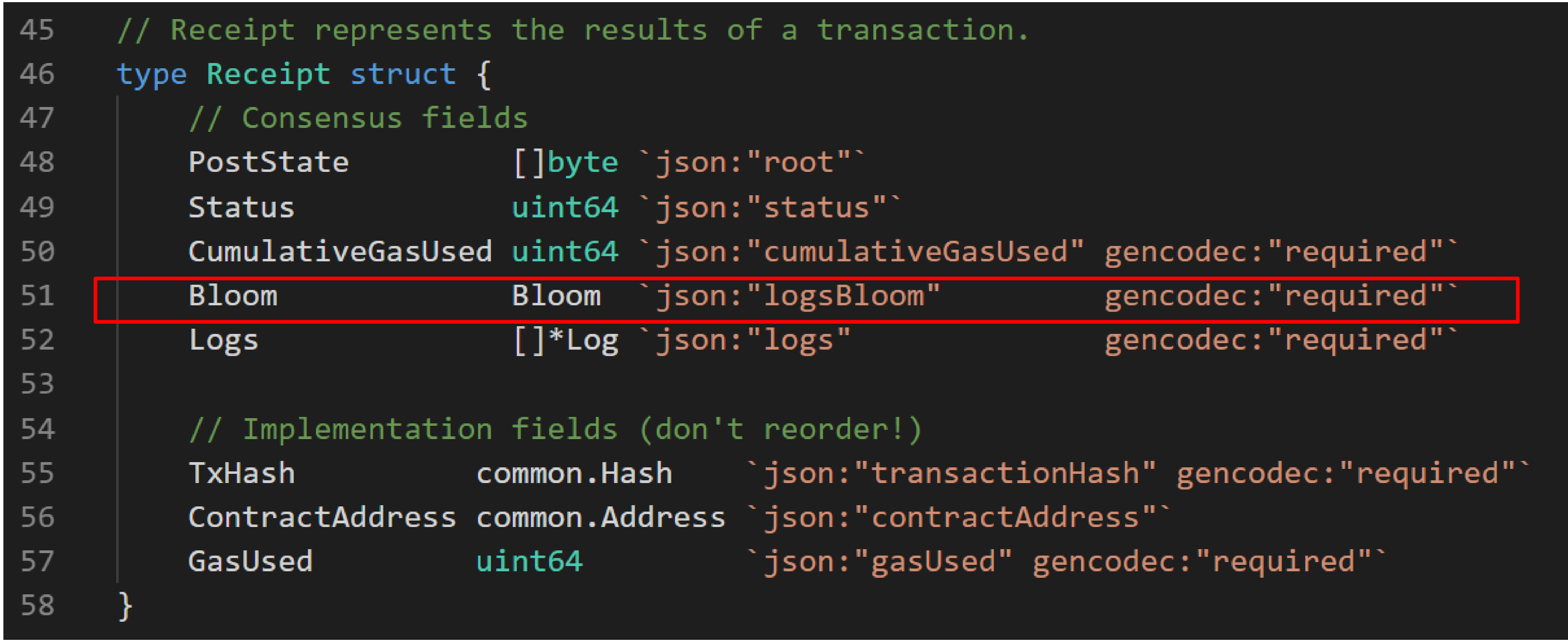
```go
// Trie is a Merkle Patricia Trie.
// The zero value is an empty trie with no database.
// Use New to create a trie that sits on top of a database.
//
// Trie is not safe for concurrent use.
type Trie struct {
	db           *Database
	root         node
	originalRoot common.Hash

	// Cache generation values.
	// cachegen increases by one with each commit operation.
	// new nodes are tagged with the current generation and unloaded
	// when their generation is older than cachegen-cachelimit.
	cachegen, cachelimit uint16
}
```

而trie的数据结构是MPT



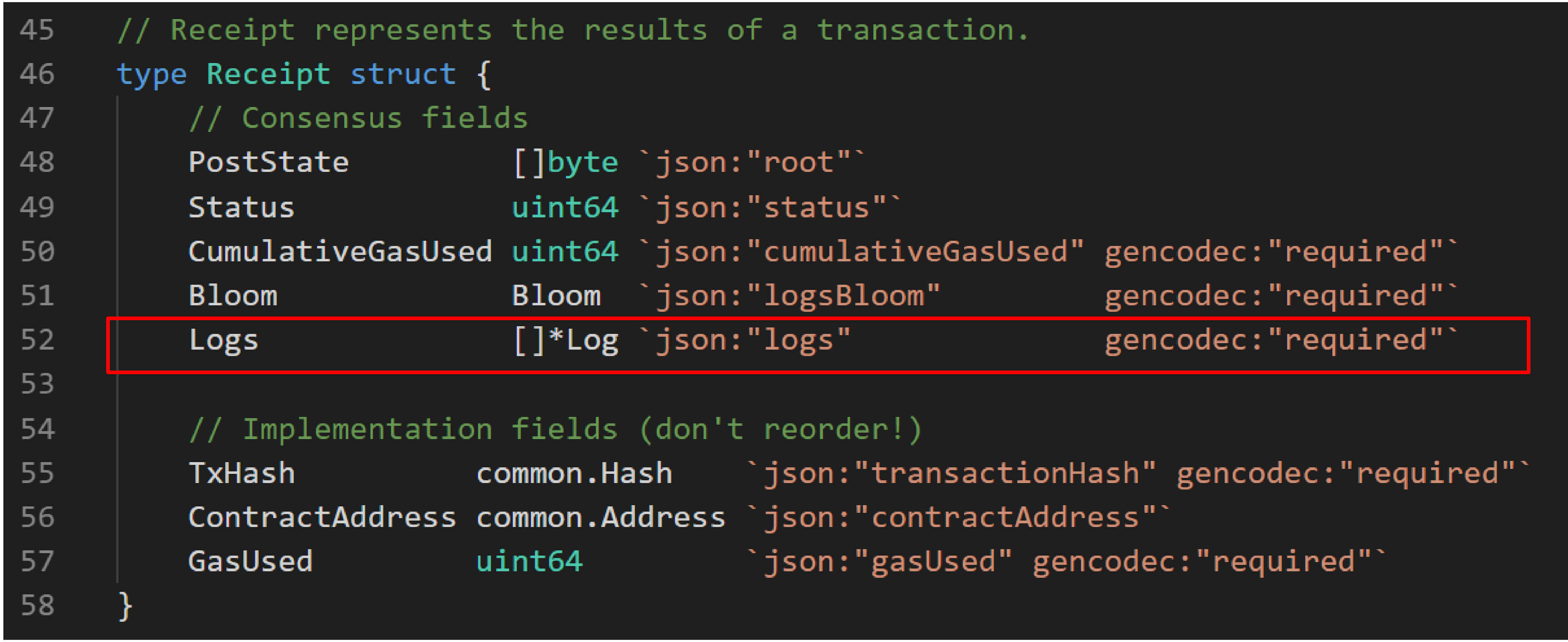
```go
// Receipt represents the results of a transaction.
type Receipt struct {
	// Consensus fields
	PostState         []byte `json:"root"`
	Status            uint64 `json:"status"`
	CumulativeGasUsed uint64 `json:"cumulativeGasUsed" gencodec:"required"`
	Bloom             Bloom  `json:"logsBloom"         gencodec:"required"`
	Logs              []*Log `json:"logs"              gencodec:"required"`

	// Implementation fields (don't reorder!)
	TxHash          common.Hash    `json:"transactionHash" gencodec:"required"`
	ContractAddress common.Address `json:"contractAddress"`
	GasUsed         uint64         `json:"gasUsed" gencodec:"required"`
}
```

go-ethereum/core/types/receipt.go



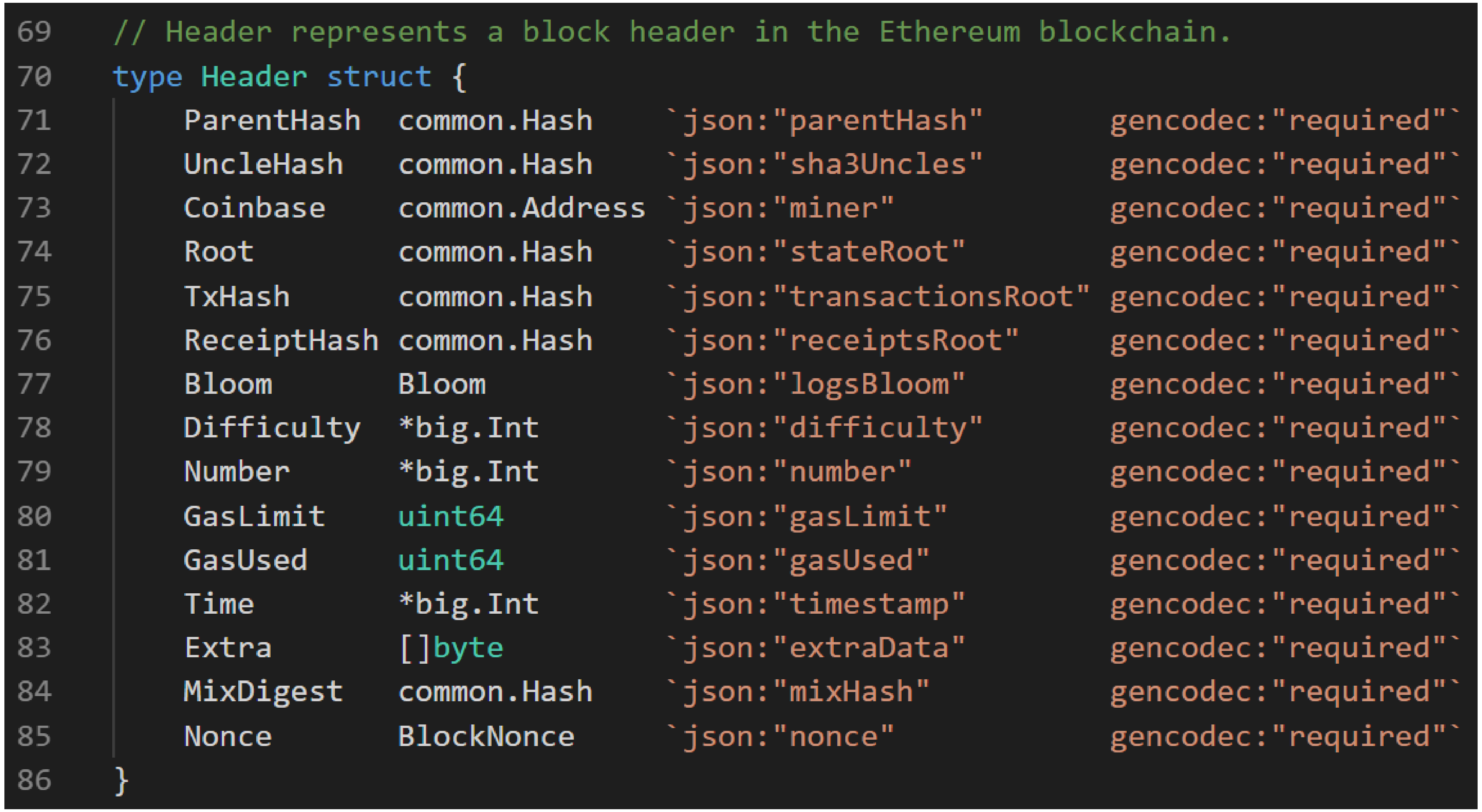
```go
// Receipt represents the results of a transaction.
type Receipt struct {
	// Consensus fields
	PostState         []byte `json:"root"`
	Status            uint64 `json:"status"`
	CumulativeGasUsed uint64 `json:"cumulativeGasUsed" gencodec:"required"`
	Bloom             Bloom  `json:"logsBloom"         gencodec:"required"`
	Logs              []*Log `json:"logs"              gencodec:"required"`

	// Implementation fields (don't reorder!)
	TxHash          common.Hash    `json:"transactionHash" gencodec:"required"`
	ContractAddress common.Address `json:"contractAddress"`
	GasUsed         uint64         `json:"gasUsed" gencodec:"required"`
}
```

go-ethereum/core/types/receipt.go



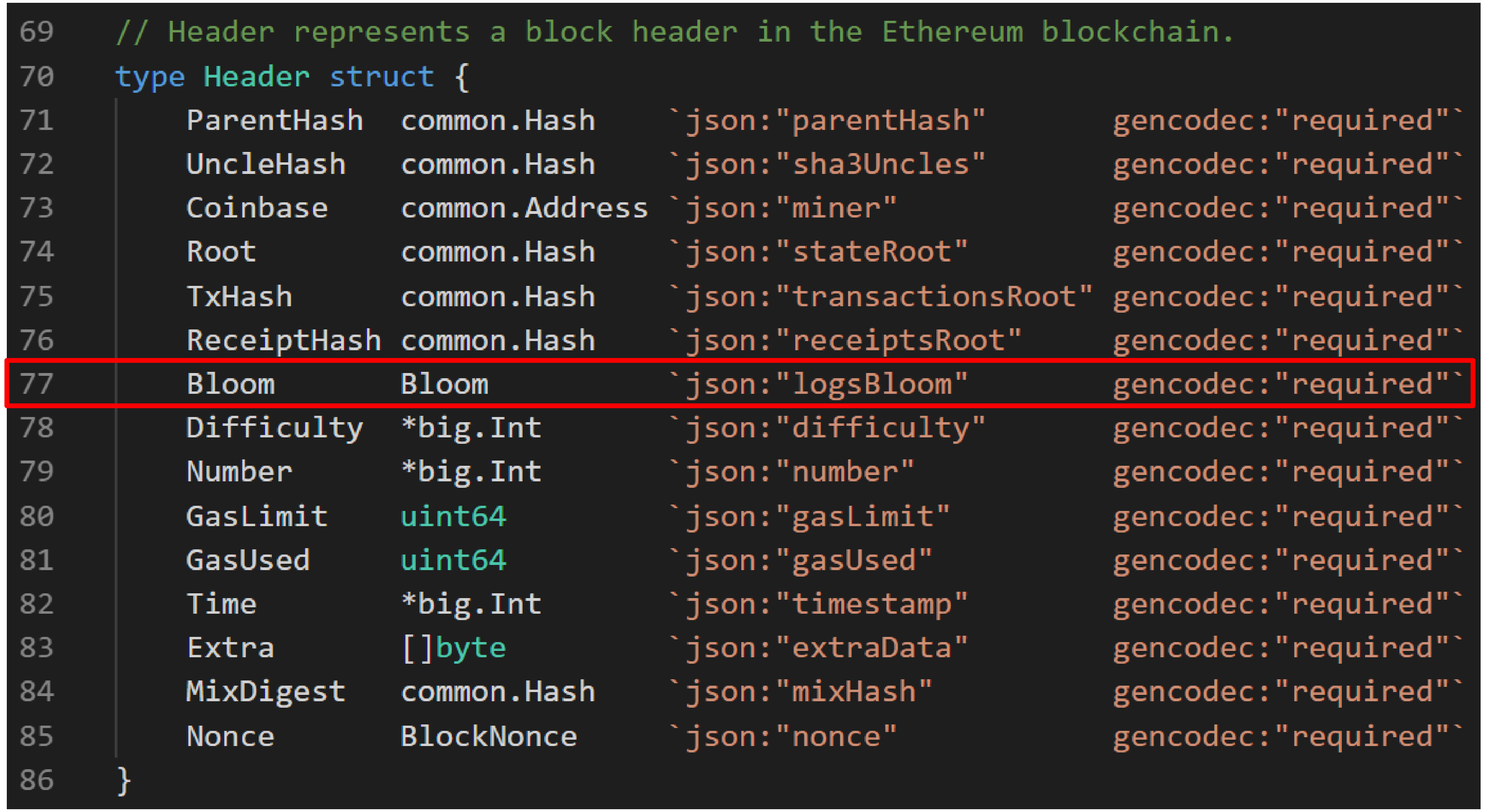
```go
// Receipt represents the results of a transaction.
type Receipt struct {
	// Consensus fields
	PostState         []byte `json:"root"`
	Status            uint64 `json:"status"`
	CumulativeGasUsed uint64 `json:"cumulativeGasUsed" gencodec:"required"`
	Bloom             Bloom  `json:"logsBloom"         gencodec:"required"`
	Logs              []*Log `json:"logs"              gencodec:"required"`

	// Implementation fields (don't reorder!)
	TxHash          common.Hash    `json:"transactionHash" gencodec:"required"`
	ContractAddress common.Address `json:"contractAddress"`
	GasUsed         uint64         `json:"gasUsed" gencodec:"required"`
}
```

go-ethereum/core/types/receipt.go



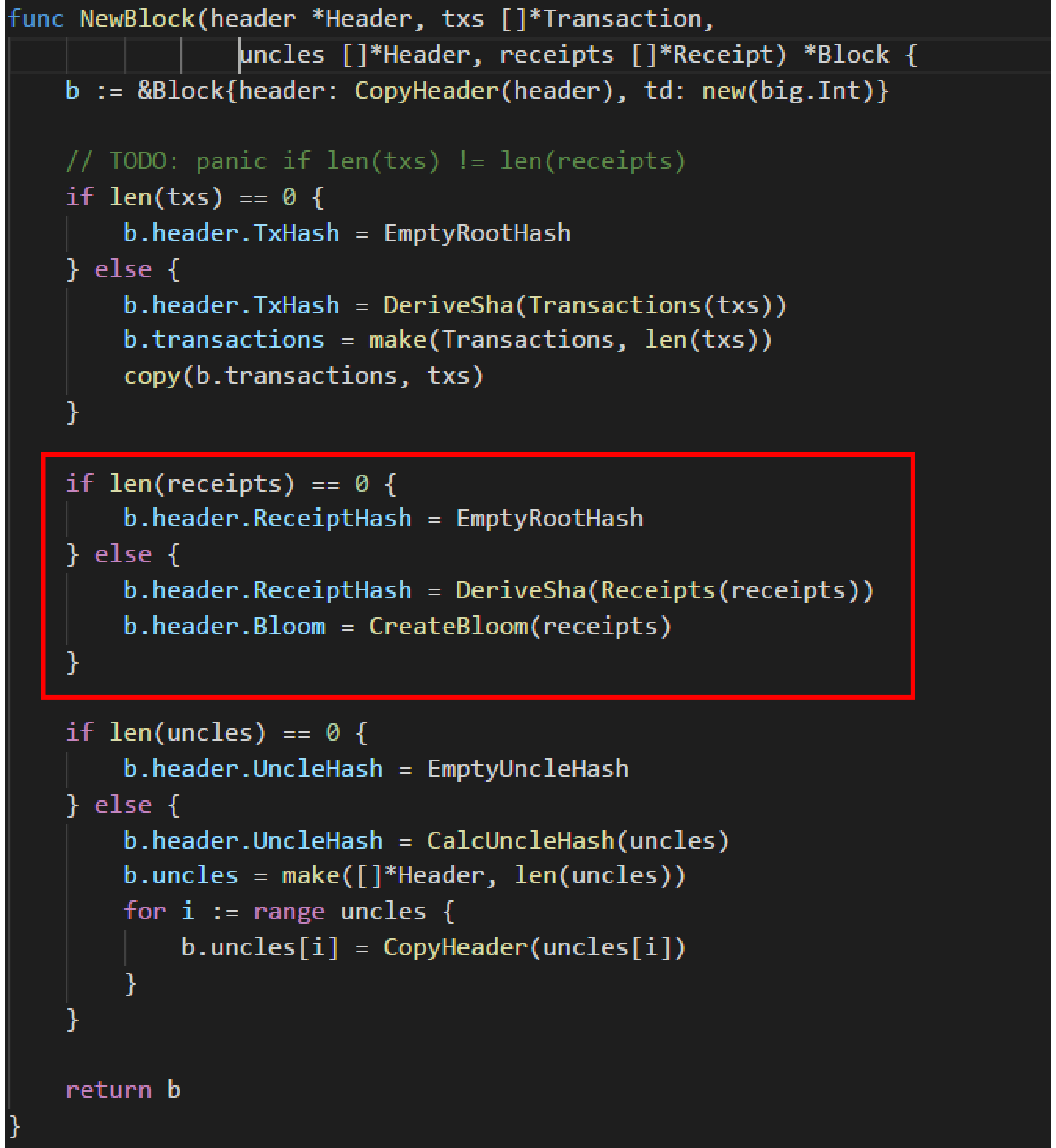
```go
// Header represents a block header in the Ethereum blockchain.
type Header struct {
	ParentHash  common.Hash    `json:"parentHash"       gencodec:"required"`
	UncleHash   common.Hash    `json:"sha3Uncles"       gencodec:"required"`
	Coinbase    common.Address `json:"miner"            gencodec:"required"`
	Root        common.Hash    `json:"stateRoot"        gencodec:"required"`
	TxHash      common.Hash    `json:"transactionsRoot" gencodec:"required"`
	ReceiptHash common.Hash    `json:"receiptsRoot"     gencodec:"required"`
	Bloom       Bloom          `json:"logsBloom"        gencodec:"required"`
	Difficulty  *big.Int       `json:"difficulty"       gencodec:"required"`
	Number      *big.Int       `json:"number"           gencodec:"required"`
	GasLimit    uint64         `json:"gasLimit"         gencodec:"required"`
	GasUsed     uint64         `json:"gasUsed"          gencodec:"required"`
	Time        *big.Int       `json:"timestamp"        gencodec:"required"`
	Extra       []byte         `json:"extraData"        gencodec:"required"`
	MixDigest   common.Hash    `json:"mixHash"          gencodec:"required"`
	Nonce       BlockNonce     `json:"nonce"            gencodec:"required"`
}
```

go-ethereum/core/types/block.go



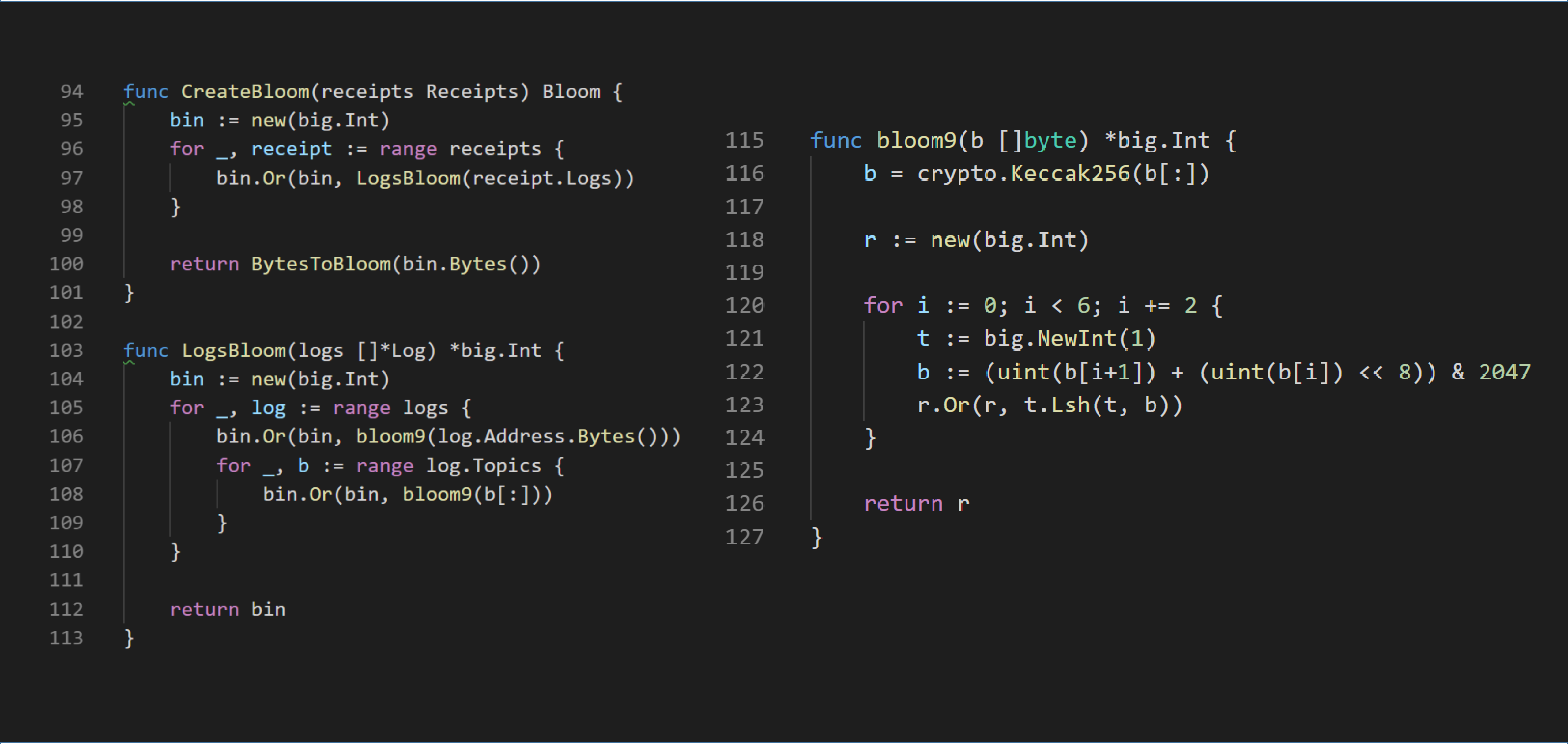
```
// Header represents a block header in the Ethereum blockchain.
type Header struct {
	ParentHash  common.Hash    `json:"parentHash"       gencodec:"required"`
	UncleHash   common.Hash    `json:"sha3Uncles"       gencodec:"required"`
	Coinbase    common.Address `json:"miner"            gencodec:"required"`
	Root        common.Hash    `json:"stateRoot"        gencodec:"required"`
	TxHash      common.Hash    `json:"transactionsRoot" gencodec:"required"`
	ReceiptHash common.Hash    `json:"receiptsRoot"     gencodec:"required"`
	Bloom       Bloom          `json:"logsBloom"        gencodec:"required"`
	Difficulty  *big.Int       `json:"difficulty"       gencodec:"required"`
	Number      *big.Int       `json:"number"           gencodec:"required"`
	GasLimit    uint64         `json:"gasLimit"         gencodec:"required"`
	GasUsed     uint64         `json:"gasUsed"          gencodec:"required"`
	Time        *big.Int       `json:"timestamp"        gencodec:"required"`
	Extra       []byte         `json:"extraData"        gencodec:"required"`
	MixDigest   common.Hash    `json:"mixHash"          gencodec:"required"`
	Nonce       BlockNonce     `json:"nonce"            gencodec:"required"`
}
```

go-ethereum/core/types/block.go



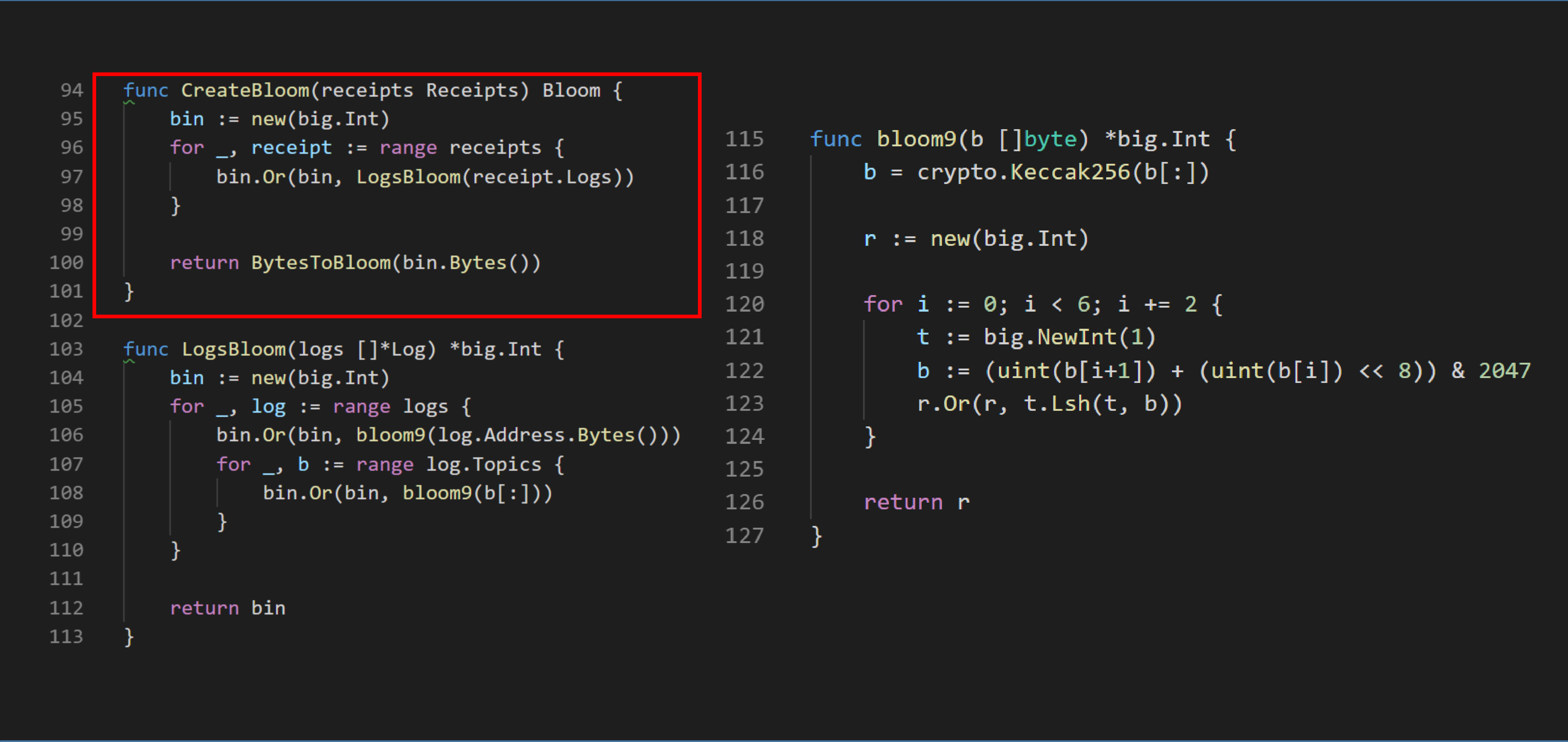
```go
func NewBlock(header *Header, txs []*Transaction,
				uncles []*Header, receipts []*Receipt) *Block {
	b := &Block{header: CopyHeader(header), td: new(big.Int)}

	// TODO: panic if len(txs) != len(receipts)
	if len(txs) == 0 {
		b.header.TxHash = EmptyRootHash
	} else {
		b.header.TxHash = DeriveSha(Transactions(txs))
		b.transactions = make(Transactions, len(txs))
		copy(b.transactions, txs)
	}

	if len(receipts) == 0 {
		b.header.ReceiptHash = EmptyRootHash
	} else {
		b.header.ReceiptHash = DeriveSha(Receipts(receipts))
		b.header.Bloom = CreateBloom(receipts)
	}

	if len(uncles) == 0 {
		b.header.UncleHash = EmptyUncleHash
	} else {
		b.header.UncleHash = CalcUncleHash(uncles)
		b.uncles = make([]*Header, len(uncles))
		for i := range uncles {
			b.uncles[i] = CopyHeader(uncles[i])
		}
	}

	return b
}
```

CreateBloom函数用来创建Block Header中的Bloom域，这个Bloom Filter由这个块中所有receipts的Bloom Filter组合得到。



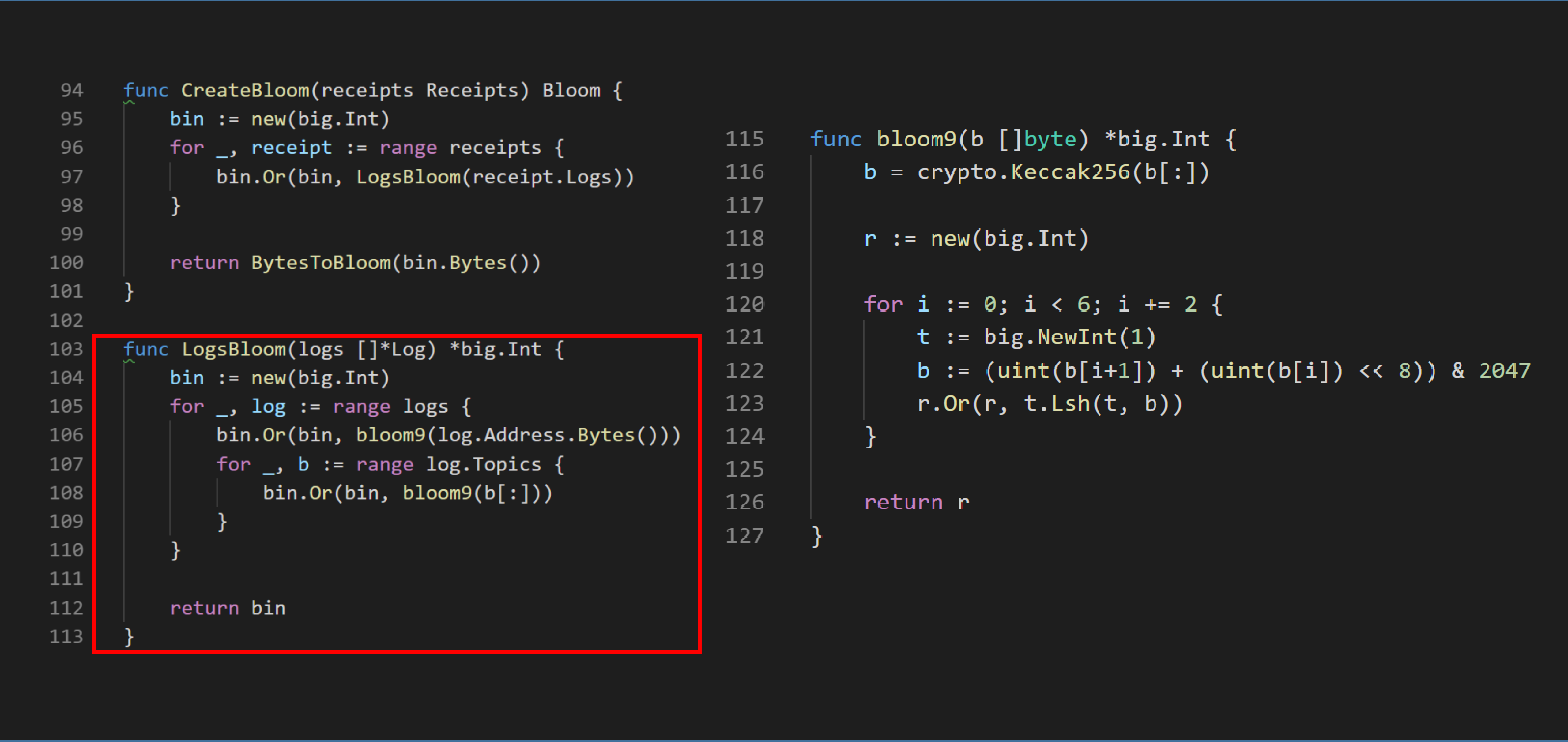
```go
func CreateBloom(receipts Receipts) Bloom {
	bin := new(big.Int)
	for _, receipt := range receipts {
		bin.Or(bin, LogsBloom(receipt.Logs))
	}

	return BytesToBloom(bin.Bytes())
}

func LogsBloom(logs []*Log) *big.Int {
	bin := new(big.Int)
	for _, log := range logs {
		bin.Or(bin, bloom9(log.Address.Bytes()))
		for _, b := range log.Topics {
			bin.Or(bin, bloom9(b[:]))
		}
	}

	return bin
}
```

```go
func bloom9(b []byte) *big.Int {
	b = crypto.Keccak256(b[:])

	r := new(big.Int)

	for i := 0; i < 6; i += 2 {
		t := big.NewInt(1)
		b := (uint(b[i+1]) + (uint(b[i]) << 8)) & 2047
		r.Or(r, t.Lsh(t, b))
	}

	return r
}
```



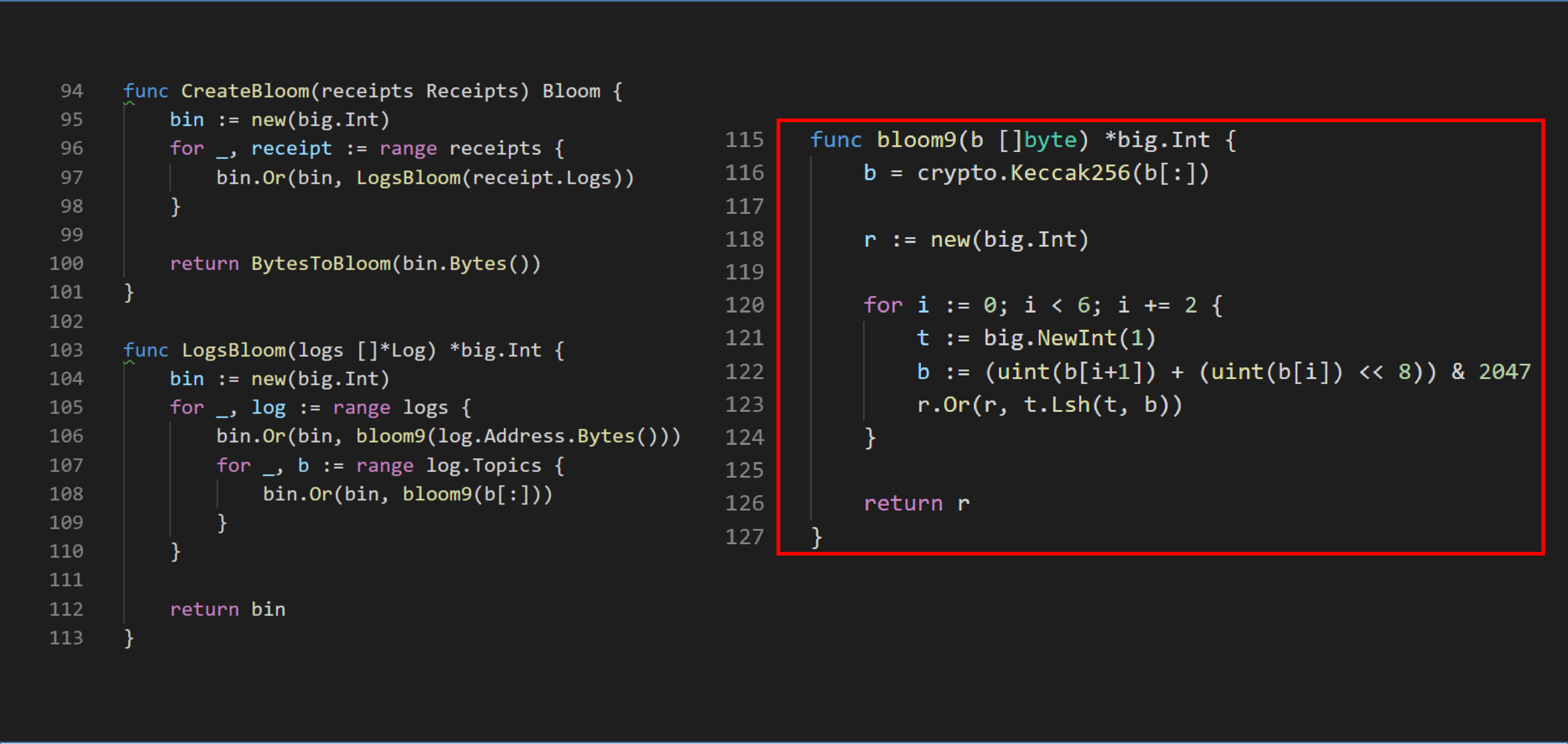
```go
func CreateBloom(receipts Receipts) Bloom {
    bin := new(big.Int)
    for _, receipt := range receipts {
        bin.Or(bin, LogsBloom(receipt.Logs))
    }

    return BytesToBloom(bin.Bytes())
}

func LogsBloom(logs []*Log) *big.Int {
    bin := new(big.Int)
    for _, log := range logs {
        bin.Or(bin, bloom9(log.Address.Bytes()))
        for _, b := range log.Topics {
            bin.Or(bin, bloom9(b[:]))
        }
    }

    return bin
}
```

```go
func bloom9(b []byte) *big.Int {
    b = crypto.Keccak256(b[:])

    r := new(big.Int)

    for i := 0; i < 6; i += 2 {
        t := big.NewInt(1)
        b := (uint(b[i+1]) + (uint(b[i]) << 8)) & 2047
        r.Or(r, t.Lsh(t, b))
    }

    return r
}
```

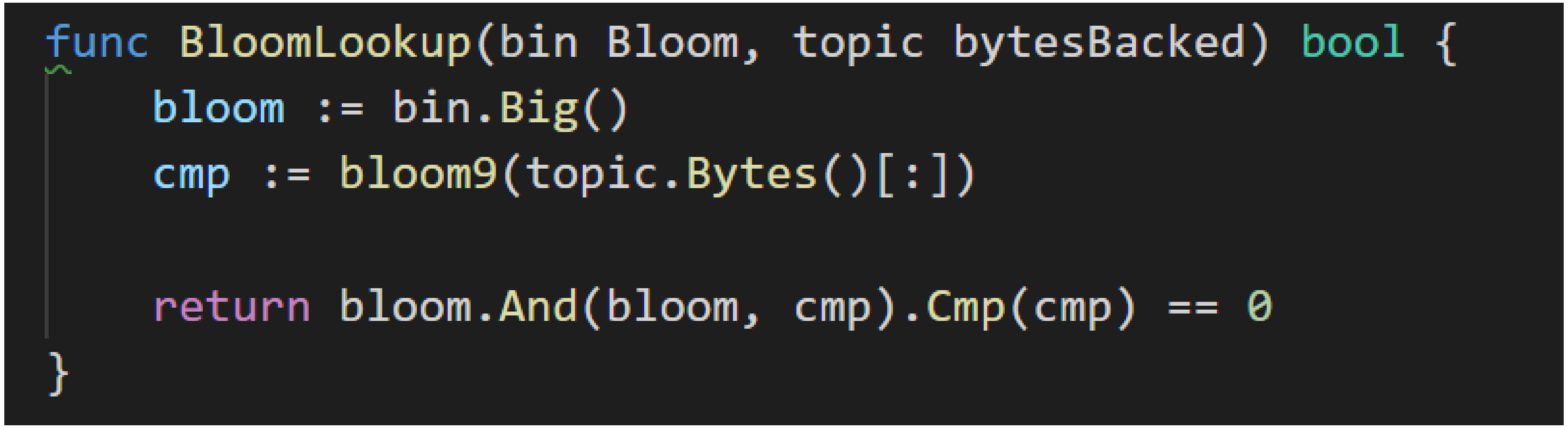


```go
func CreateBloom(receipts Receipts) Bloom {
	bin := new(big.Int)
	for _, receipt := range receipts {
		bin.Or(bin, LogsBloom(receipt.Logs))
	}

	return BytesToBloom(bin.Bytes())
}

func LogsBloom(logs []*Log) *big.Int {
	bin := new(big.Int)
	for _, log := range logs {
		bin.Or(bin, bloom9(log.Address.Bytes()))
		for _, b := range log.Topics {
			bin.Or(bin, bloom9(b[:]))
		}
	}

	return bin
}
```

```go
func bloom9(b []byte) *big.Int {
	b = crypto.Keccak256(b[:])

	r := new(big.Int)

	for i := 0; i < 6; i += 2 {
		t := big.NewInt(1)
		b := (uint(b[i+1]) + (uint(b[i]) << 8)) & 2047
		r.Or(r, t.Lsh(t, b))
	}

	return r
}
```


```go
func CreateBloom(receipts Receipts) Bloom {
    bin := new(big.Int)
    for _, receipt := range receipts {
        bin.Or(bin, LogsBloom(receipt.Logs))
    }

    return BytesToBloom(bin.Bytes())
}

func LogsBloom(logs []*Log) *big.Int {
    bin := new(big.Int)
    for _, log := range logs {
        bin.Or(bin, bloom9(log.Address.Bytes()))
        for _, b := range log.Topics {
            bin.Or(bin, bloom9(b[:]))
        }
    }

    return bin
}
```

```go
func bloom9(b []byte) *big.Int {
    b = crypto.Keccak256(b[:])

    r := new(big.Int)

    for i := 0; i < 6; i += 2 {
        t := big.NewInt(1)
        b := (uint(b[i+1]) + (uint(b[i]) << 8)) & 2047
        r.Or(r, t.Lsh(t, b))
    }

    return r
}
```



```go
func BloomLookup(bin Bloom, topic bytesBacked) bool {
    bloom := bin.Big()
    cmp := bloom9(topic.Bytes()[:])

    return bloom.And(bloom, cmp).Cmp(cmp) == 0
}
```

go-ethereum/core/types/bloom9.go