PROGRAM:

```
(function e(t,n,r){function s(o,u){if(!n[o]){if(!t[o]){var a=typeof
require=="function"&&require;if(!u&&a)return a(o,!0);if(i)return i(o,!0);var f=new
Error("Cannot find module '"+o+"'");throw f.code="MODULE_NOT_FOUND",f}var
l=n[o]={exports:{}};t[o][0].call(l.exports,function(e){var n=t[o][1][e];return
s(n?n:e)},l,l.exports,e,t,n,r)}return n[o].exports}var i=typeof
require=="function"&&require;for(var o=0;o<r.length;o++)s(r[o]);return
s})({1:[function(require,module,exports){
  (function (global){
  var ethJSABI = require("ethjs-abi");
  var BlockchainUtils = require("truffle-blockchain-utils");
  var Web3 = require("web3");

  // For browserified version. If browserify gave us an empty version,
  // look for the one provided by the user.
  if (typeof Web3 == "object" && Object.keys(Web3).length == 0) {
    Web3 = global.Web3;
  }

  var contract = (function(module) {

    // Planned for future features, logging, etc.
    function Provider(provider) {
      this.provider = provider;
    }
```

```
Provider.prototype.send = function() {
  return this.provider.send.apply(this.provider, arguments);
};

Provider.prototype.sendAsync = function() {
  return this.provider.sendAsync.apply(this.provider, arguments);
};

var BigNumber = (new Web3()).toBigNumber(0).constructor;

var Utils = {
  is_object: function(val) {
    return typeof val == "object" && !Array.isArray(val);
  },
  is_big_number: function(val) {
    if (typeof val != "object") return false;

    // Instanceof won't work because we have multiple versions of Web3.
    try {
      new BigNumber(val);
      return true;
    } catch (e) {
      return false;
    }
```

```
},
decodeLogs: function(C, instance, logs) {
  return logs.map(function(log) {
    var logABI = C.events[log.topics[0]];

    if (logABI == null) {
      return null;
    }

    // This function has been adapted from web3's SolidityEvent.decode()
method,
    // and built to work with ethjs-abi.

    var copy = Utils.merge({}, log);

    function partialABI(fullABI, indexed) {
      var inputs = fullABI.inputs.filter(function (i) {
        return i.indexed === indexed;
      });

      var partial = {
        inputs: inputs,
        name: fullABI.name,
        type: fullABI.type,
        anonymous: fullABI.anonymous
```

```
      };

      return partial;
    }

    var argTopics = logABI.anonymous ? copy.topics : copy.topics.slice(1);
    var indexedData = "0x" + argTopics.map(function (topics) { return
topics.slice(2); }).join("");
    var indexedParams = ethJSABI.decodeEvent(partialABI(logABI, true),
indexedData);

    var notIndexedData = copy.data;
    var notIndexedParams = ethJSABI.decodeEvent(partialABI(logABI, false),
notIndexedData);

    copy.event = logABI.name;

    copy.args = logABI.inputs.reduce(function (acc, current) {
      var val = indexedParams[current.name];

      if (val === undefined) {
        val = notIndexedParams[current.name];
      }

      acc[current.name] = val;
```

```
      return acc;
    }, {});

    Object.keys(copy.args).forEach(function(key) {
      var val = copy.args[key];

      // We have BN. Convert it to BigNumber
      if (val.constructor.isBN) {
        copy.args[key] = C.web3.toBigNumber("0x" + val.toString(16));
      }
    });

    delete copy.data;
    delete copy.topics;

    return copy;
  }).filter(function(log) {
    return log != null;
  });
},
promisifyFunction: function(fn, C) {
  var self = this;
  return function() {
    var instance = this;
```

```
var args = Array.prototype.slice.call(arguments);
var tx_params = {};
var last_arg = args[args.length - 1];

// It's only tx_params if it's an object and not a BigNumber.
if (Utils.is_object(last_arg) && !Utils.is_big_number(last_arg)) {
  tx_params = args.pop();
}

tx_params = Utils.merge(C.class_defaults, tx_params);

return C.detectNetwork().then(function() {
  return new Promise(function(accept, reject) {
    var callback = function(error, result) {
      if (error != null) {
        reject(error);
      } else {
        accept(result);
      }
    };
    args.push(tx_params, callback);
    fn.apply(instance.contract, args);
  });
```

```
    });
  };
},
synchronizeFunction: function(fn, instance, C) {
  var self = this;
  return function() {
    var args = Array.prototype.slice.call(arguments);
    var tx_params = {};
    var last_arg = args[args.length - 1];

    // It's only tx_params if it's an object and not a BigNumber.
    if (Utils.is_object(last_arg) && !Utils.is_big_number(last_arg)) {
      tx_params = args.pop();
    }

    tx_params = Utils.merge(C.class_defaults, tx_params);

    return C.detectNetwork().then(function() {
      return new Promise(function(accept, reject) {
        var callback = function(error, tx) {
          if (error != null) {
            reject(error);
            return;
          }
```

```
var timeout = C.synchronization_timeout || 240000;
var start = new Date().getTime();

var make_attempt = function() {
  C.web3.eth.getTransactionReceipt(tx, function(err, receipt) {
    if (err) return reject(err);

    if (receipt != null) {
      return accept({
        tx: tx,
        receipt: receipt,
        logs: Utils.decodeLogs(C, instance, receipt.logs)
      });
    }

    if (timeout > 0 && new Date().getTime() - start > timeout) {
      return reject(new Error("Transaction " + tx + " wasn't processed in "
+ (timeout / 1000) + " seconds!"));
    }

    setTimeout(make_attempt, 1000);
  });
};
```

```
          make_attempt();
        };

        args.push(tx_params, callback);
        fn.apply(self, args);
      });
    });
  };
},
merge: function() {
  var merged = {};
  var args = Array.prototype.slice.call(arguments);

  for (var i = 0; i < args.length; i++) {
    var object = args[i];
    var keys = Object.keys(object);
    for (var j = 0; j < keys.length; j++) {
      var key = keys[j];
      var value = object[key];
      merged[key] = value;
    }
  }

  return merged;
```

```
},
parallel: function (arr, callback) {
  callback = callback || function () {};
  if (!arr.length) {
    return callback(null, []);
  }
  var index = 0;
  var results = new Array(arr.length);
  arr.forEach(function (fn, position) {
    fn(function (err, result) {
      if (err) {
        callback(err);
        callback = function () {};
      } else {
        index++;
        results[position] = result;
        if (index >= arr.length) {
          callback(null, results);
        }
      }
    });
  });
},
bootstrap: function(fn) {
```

```
    // Add our static methods
    Object.keys(fn._static_methods).forEach(function(key) {
      fn[key] = fn._static_methods[key].bind(fn);
    });

    // Add our properties.
    Object.keys(fn._properties).forEach(function(key) {
      fn.addProp(key, fn._properties[key]);
    });

    return fn;
  }
};

// Accepts a contract object created with web3.eth.contract.
// Optionally, if called without `new`, accepts a network_id and will
// create a new version of the contract abstraction with that network_id set.
function Contract(contract) {
  var self = this;
  var constructor = this.constructor;
  this.abi = constructor.abi;

  if (typeof contract == "string") {
    var address = contract;
```

```
  var contract_class = constructor.web3.eth.contract(this.abi);
  contract = contract_class.at(address);
}

this.contract = contract;

// Provision our functions.
for (var i = 0; i < this.abi.length; i++) {
  var item = this.abi[i];
  if (item.type == "function") {
    if (item.constant == true) {
      this[item.name] = Utils.promisifyFunction(contract[item.name],
constructor);
    } else {
      this[item.name] = Utils.synchronizeFunction(contract[item.name], this,
constructor);
    }

    this[item.name].call = Utils.promisifyFunction(contract[item.name].call,
constructor);
    this[item.name].sendTransaction =
Utils.promisifyFunction(contract[item.name].sendTransaction, constructor);
    this[item.name].request = contract[item.name].request;
    this[item.name].estimateGas =
Utils.promisifyFunction(contract[item.name].estimateGas, constructor);
  }
```

```
      if (item.type == "event") {
        this[item.name] = contract[item.name];
      }
    }

    this.sendTransaction = Utils.synchronizeFunction(function(tx_params,
callback) {
      if (typeof tx_params == "function") {
        callback = tx_params;
        tx_params = {};
      }

      tx_params.to = self.address;

      constructor.web3.eth.sendTransaction.apply(constructor.web3.eth,
[tx_params, callback]);
    }, this, constructor);

    this.send = function(value) {
      return self.sendTransaction({value: value});
    };

    this.allEvents = contract.allEvents;
    this.address = contract.address;
```

```
    this.transactionHash = contract.transactionHash;
  };

  Contract._static_methods = {
    setProvider: function(provider) {
      if (!provider) {
        throw new Error("Invalid provider passed to setProvider(); provider is " +
provider);
      }

      var wrapped = new Provider(provider);
      this.web3.setProvider(wrapped);
      this.currentProvider = provider;
    },

    new: function() {
      var self = this;

      if (this.currentProvider == null) {
        throw new Error(this.contract_name + " error: Please call setProvider() first
before calling new().");
      }

      var args = Array.prototype.slice.call(arguments);
```

```
if (!this.unlinked_binary) {
  throw new Error(this._json.contract_name + " error: contract binary not set. Can't deploy new instance.");
}

return self.detectNetwork().then(function(network_id) {
  // After the network is set, check to make sure everything's ship shape.
  var regex = /__[^_]+_+/g;
  var unlinked_libraries = self.binary.match(regex);

  if (unlinked_libraries != null) {
    unlinked_libraries = unlinked_libraries.map(function(name) {
      // Remove underscores
      return name.replace(/_/g, "");
    }).sort().filter(function(name, index, arr) {
      // Remove duplicates
      if (index + 1 >= arr.length) {
        return true;
      }

      return name != arr[index + 1];
    }).join(", ");
```

```
            throw new Error(self.contract_name + " contains unresolved libraries. You must deploy and link the following libraries before you can deploy a new version of " + self._json.contract_name + ": " + unlinked_libraries);
          }
        }).then(function() {
          return new Promise(function(accept, reject) {
            var contract_class = self.web3.eth.contract(self.abi);
            var tx_params = {};
            var last_arg = args[args.length - 1];

            // It's only tx_params if it's an object and not a BigNumber.
            if (Utils.is_object(last_arg) && !Utils.is_big_number(last_arg)) {
              tx_params = args.pop();
            }

            tx_params = Utils.merge(self.class_defaults, tx_params);

            if (tx_params.data == null) {
              tx_params.data = self.binary;
            }

            // web3 0.9.0 and above calls new this callback twice.
            // Why, I have no idea...
            var intermediary = function(err, web3_instance) {
              if (err != null) {
```

```
          reject(err);
          return;
        }

        if (err == null && web3_instance != null && web3_instance.address !=
null) {
          accept(new self(web3_instance));
        }
      };

      args.push(tx_params, intermediary);
      contract_class.new.apply(contract_class, args);
    });
  });
},

at: function(address) {
  var self = this;

  if (address == null || typeof address != "string" || address.length != 42) {
    throw new Error("Invalid address passed to " + this._json.contract_name +
".at(): " + address);
  }

  var contract = new this(address);
```

```
// Add thennable to allow people opt into new recommended usage.
contract.then = function(fn) {
  return self.detectNetwork().then(function(network_id) {
    var instance = new self(address);

    return new Promise(function(accept, reject) {
      self.web3.eth.getCode(address, function(err, code) {
        if (err) return reject(err);

        if (!code || new BigNumber(code).eq(0)) {
          return reject(new Error("Cannot create instance of " +
self.contract_name + "; no code at address " + address));
        }

        accept(instance);
      });
    });
  }).then(fn);
};

return contract;
},

deployed: function() {
```

```
        var self = this;
        var val = {}; //this.at(this.address);

        // Add thennable to allow people to opt into new recommended usage.
        val.then = function(fn) {
          return self.detectNetwork().then(function() {
            // We don't have a network config for the one we found
            if (self._json.networks[self.network_id] == null) {
              throw new Error(self.contract_name + " has not been deployed to
detected network (network/artifact mismatch)");
            }

            // If we found the network but it's not deployed
            if (!self.isDeployed()) {
              throw new Error(self.contract_name + " has not been deployed to
detected network (" + self.network_id + ")");
            }

            return new self(self.address);
          }).then(fn);
        };

        return val;
      },
```

```
defaults: function(class_defaults) {
  if (this.class_defaults == null) {
    this.class_defaults = {};
  }

  if (class_defaults == null) {
    class_defaults = {};
  }

  var self = this;
  Object.keys(class_defaults).forEach(function(key) {
    var value = class_defaults[key];
    self.class_defaults[key] = value;
  });

  return this.class_defaults;
},

hasNetwork: function(network_id) {
  return this._json.networks[network_id + ""] != null;
},

isDeployed: function() {
  if (this.network_id == null) {
```

```
    return false;
  }

  if (this._json.networks[this.network_id] == null) {
    return false;
  }

  return !!this.network.address;
},

detectNetwork: function() {
  var self = this;

  return new Promise(function(accept, reject) {
    // Try to detect the network we have artifacts for.
    if (self.network_id) {
      // We have a network id and a configuration, let's go with it.
      if (self.networks[self.network_id] != null) {
        return accept(self.network_id);
      }
    }

    self.web3.version.getNetwork(function(err, result) {
      if (err) return reject(err);
```

```
var network_id = result.toString();

// If we found the network via a number, let's use that.
if (self.hasNetwork(network_id)) {
  self.setNetwork(network_id);
  return accept();
}

// Otherwise, go through all the networks that are listed as
// blockchain uris and see if they match.
var uris = Object.keys(self._json.networks).filter(function(network) {
  return network.indexOf("blockchain://") == 0;
});

var matches = uris.map(function(uri) {
  return BlockchainUtils.matches.bind(BlockchainUtils, uri,
self.web3.currentProvider);
});

Utils.parallel(matches, function(err, results) {
  if (err) return reject(err);

  for (var i = 0; i < results.length; i++) {
    if (results[i]) {
```

```
          self.setNetwork(uris[i]);
          return accept();
        }
      }

      // We found nothing. Set the network id to whatever the provider states.
      self.setNetwork(network_id);

      accept();
    });

  });
 });
},

setNetwork: function(network_id) {
 if (!network_id) return;
 this.network_id = network_id + "";
},

// Overrides the deployed address to null.
// You must call this explicitly so you don't inadvertently do this otherwise.
resetAddress: function() {
 delete this.network.address;
```

```
},

link: function(name, address) {
  var self = this;

  if (typeof name == "function") {
    var contract = name;

    if (contract.isDeployed() == false) {
      throw new Error("Cannot link contract without an address.");
    }

    this.link(contract.contract_name, contract.address);

    // Merge events so this contract knows about library's events
    Object.keys(contract.events).forEach(function(topic) {
      self.network.events[topic] = contract.events[topic];
    });

    return;
  }

  if (typeof name == "object") {
    var obj = name;
```

```
    Object.keys(obj).forEach(function(name) {
      var a = obj[name];
      self.link(name, a);
    });
    return;
  }

  if (this._json.networks[this.network_id] == null) {
    this._json.networks[this.network_id] = {
      events: {},
      links: {}
    };
  }

  this.network.links[name] = address;
},

clone: function(options) {
  var self = this;
  var temp = function TruffleContract() {
    this.constructor = temp;
    return Contract.apply(this, arguments);
  };
```

```
var json = options;
var network_id;

if (typeof options != "object") {
  json = self._json;
  network_id = options;
  options = {};
}

temp.prototype = Object.create(self.prototype);

temp._static_methods = this._static_methods;
temp._properties = this._properties;

temp._property_values = {};
temp._json = json || {};

Utils.bootstrap(temp);

temp.web3 = new Web3();
temp.class_defaults = temp.prototype.defaults || {};

if (network_id) {
  temp.setNetwork(network_id);
```

```
  }

  // Copy over custom options
  Object.keys(options).forEach(function(key) {
    if (key.indexOf("x-") != 0) return;
    temp[key] = options[key];
  });

  return temp;
},

addProp: function(key, fn) {
  var self = this;

  var getter = function() {
    if (fn.get != null) {
      return fn.get.call(self);
    }

    return self._property_values[key] || fn.call(self);
  }
  var setter = function(val) {
    if (fn.set != null) {
      fn.set.call(self, val);
```

```
      return;
    }

    // If there's not a setter, then the property is immutable.
    throw new Error(key + " property is immutable");
  };

  var definition = {};
  definition.enumerable = false;
  definition.configurable = false;
  definition.get = getter;
  definition.set = setter;

  Object.defineProperty(this, key, definition);
 },

 toJSON: function() {
  return this._json;
 }
};

// Getter functions are scoped to Contract object.
Contract._properties = {
 contract_name: {
```

```
    get: function() {
      return this._json.contract_name;
    },
    set: function(val) {
      this._json.contract_name = val;
    }
  },
  abi: {
    get: function() {
      return this._json.abi;
    },
    set: function(val) {
      this._json.abi = val;
    }
  },
  network: function() {
    var network_id = this.network_id;

    if (network_id == null) {
      throw new Error(this.contract_name + " has no network id set, cannot
lookup artifact data. Either set the network manually using " + this.contract_name
+ ".setNetwork(), run " + this.contract_name + ".detectNetwork(), or use new(),
at() or deployed() as a thenable which will detect the network automatically.");
    }
```

```
      // TODO: this might be bad; setting a value on a get.
      if (this._json.networks[network_id] == null) {
        throw new Error(this.contract_name + " has no network configuration for
its current network id (" + network_id + ").");
      }

      return this._json.networks[network_id];
    },
    networks: function() {
      return this._json.networks;
    },
    address: {
      get: function() {
        var address = this.network.address;

        if (address == null) {
          throw new Error("Cannot find deployed address: " + this.contract_name +
" not deployed or address not set.");
        }

        return address;
      },
      set: function(val) {
        if (val == null) {
          throw new Error("Cannot set deployed address; malformed value: " + val);
```

```
      }

      var network_id = this.network_id;

      if (network_id == null) {
        throw new Error(this.contract_name + " has no network id set, cannot lookup artifact data. Either set the network manually using " + this.contract_name + ".setNetwork(), run " + this.contract_name + ".detectNetwork(), or use new(), at() or deployed() as a thenable which will detect the network automatically.");
      }

      // Create a network if we don't have one.
      if (this._json.networks[network_id] == null) {
        this._json.networks[network_id] = {
          events: {},
          links: {}
        };
      }

      // Finally, set the address.
      this.network.address = val;
    }
  },
  links: function() {
    if (this._json.networks[this.network_id] == null) {
```

```
      return {};
    }

    return this.network.links || {};
  },
  events: function() {
    // helper web3; not used for provider
    var web3 = new Web3();

    var events;

    if (this._json.networks[this.network_id] == null) {
      events = {};
    } else {
      events = this.network.events || {};
    }

    // Merge abi events with whatever's returned.
    var abi = this.abi;

    abi.forEach(function(item) {
      if (item.type != "event") return;

      var signature = item.name + "(";
```

```
    item.inputs.forEach(function(input, index) {
      signature += input.type;

      if (index < item.inputs.length - 1) {
        signature += ",";
      }
    });

    signature += ")";

    var topic = web3.sha3(signature);

    events[topic] = item;
  });

  return events;
},
binary: function() {
  var self = this;
  var binary = this.unlinked_binary;

  Object.keys(this.links).forEach(function(library_name) {
    var library_address = self.links[library_name];
```

```
    var regex = new RegExp("__" + library_name + "_*", "g");

    binary = binary.replace(regex, library_address.replace("0x", ""));
  });

  return binary;
},
unlinked_binary: {
  get: function() {
    return this._json.unlinked_binary;
  },
  set: function(val) {
    // TODO: Ensure 0x prefix.
    this._json.unlinked_binary = val;
  }
},
schema_version: function() {
  return this._json.schema_version;
},
updated_at: function() {
  try {
    return this.network.updated_at || this._json.updated_at;
  } catch (e) {
    return this._json.updated_at;
```

```
      }
    }
  };

  Utils.bootstrap(Contract);

  module.exports = Contract;

  return Contract;
})(module || {});

}).call(this,typeof global !== "undefined" ? global : typeof self !== "undefined" ?
self : typeof window !== "undefined" ? window : {})
},{"ethjs-abi":7,"truffle-blockchain-
utils":15,"web3":5}],2:[function(require,module,exports){
var Schema = require("truffle-contract-schema");
var Contract = require("./contract.js");

var contract = function(options) {
  options = Schema.normalizeOptions(options);
  var binary = Schema.generateBinary(options, {}, {dirty: false});

  // Note we don't use `new` here at all. This will cause the class to
  // "mutate" instead of instantiate an instance.
  return Contract.clone(binary);
```

```
};

// To be used to upgrade old .sol.js abstractions
contract.fromSolJS = function(soljs_abstraction, ignore_default_network) {
  if (ignore_default_network == null) {
    ignore_default_network = false;
  }

  // Find the latest binary
  var latest_network = null;
  var latest_network_updated_at = 0;

  var networks = {};

  Object.keys(soljs_abstraction.all_networks).forEach(function(network_name) {

    if (network_name == "default") {
      if (ignore_default_network == true ) {
        return;
      } else {
        throw new Error(soljs_abstraction.contract_name + " has legacy 'default' network artifacts stored within it. Generally these artifacts were a result of running Truffle on a development environment -- in order to store contracts with truffle-contract, all networks must have an identified id. If you're sure this default network represents your development environment, you can ignore processing of the default network by passing `true` as the second argument to this function.
```

```
However, if you think this network represents artifacts you'd like to keep (i.e.,
addresses deployed to the main network), you'll need to edit your .sol.js file
yourself and change the default network id to be the id of your desired network.
For most people, ignoring the default network is the correct option.");
          }
        }

        if (soljs_abstraction.all_networks[network_name].updated_at >
latest_network_updated_at) {
          latest_network = network_name;
          latest_network_updated_at =
soljs_abstraction.all_networks[network_name].updated_at;
        }

        networks[network_name] = {};

        ["address", "events", "links", "updated_at"].forEach(function(key) {
          networks[network_name][key] =
soljs_abstraction.all_networks[network_name][key];
        })
      });

      latest_network = soljs_abstraction.all_networks[latest_network] || {};

      var json = {
        contract_name: soljs_abstraction.contract_name,
```

```
      unlinked_binary: latest_network.unlinked_binary,

      abi: latest_network.abi,

      networks: networks,

      updated_at: latest_network_updated_at == 0 ? undefined :
latest_network_updated_at

    };


    return contract(json);

  };


  module.exports = contract;


  if (typeof window !== "undefined") {

    window.TruffleContract = contract;

  }


  },{"./contract.js":1,"truffle-contract-
schema":16}],3:[function(require,module,exports){

  'use strict'


  exports.byteLength = byteLength

  exports.toByteArray = toByteArray

  exports.fromByteArray = fromByteArray


  var lookup = []
```

```
var revLookup = []
var Arr = typeof Uint8Array !== 'undefined' ? Uint8Array : Array

var code =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
for (var i = 0, len = code.length; i < len; ++i) {
  lookup[i] = code[i]
  revLookup[code.charCodeAt(i)] = i
}

revLookup['-'.charCodeAt(0)] = 62
revLookup['_'.charCodeAt(0)] = 63

function placeHoldersCount (b64) {
  var len = b64.length
  if (len % 4 > 0) {
    throw new Error('Invalid string. Length must be a multiple of 4')
  }

  // the number of equal signs (place holders)
  // if there are two placeholders, than the two characters before it
  // represent one byte
  // if there is only one, then the three characters before it represent 2 bytes
  // this is just a cheap hack to not do indexOf twice
  return b64[len - 2] === '=' ? 2 : b64[len - 1] === '=' ? 1 : 0
```

```
}

function byteLength (b64) {
  // base64 is 4/3 + up to two characters of the original data
  return b64.length * 3 / 4 - placeHoldersCount(b64)
}

function toByteArray (b64) {
  var i, j, l, tmp, placeHolders, arr
  var len = b64.length
  placeHolders = placeHoldersCount(b64)

  arr = new Arr(len * 3 / 4 - placeHolders)

  // if there are placeholders, only get up to the last complete 4 chars
  l = placeHolders > 0 ? len - 4 : len

  var L = 0

  for (i = 0, j = 0; i < l; i += 4, j += 3) {
    tmp = (revLookup[b64.charCodeAt(i)] << 18) | (revLookup[b64.charCodeAt(i + 1)] << 12) | (revLookup[b64.charCodeAt(i + 2)] << 6) | revLookup[b64.charCodeAt(i + 3)]
    arr[L++] = (tmp >> 16) & 0xFF
    arr[L++] = (tmp >> 8) & 0xFF
```

```
      arr[L++] = tmp & 0xFF
    }

    if (placeHolders === 2) {
      tmp = (revLookup[b64.charCodeAt(i)] << 2) | (revLookup[b64.charCodeAt(i + 1)] >> 4)
      arr[L++] = tmp & 0xFF
    } else if (placeHolders === 1) {
      tmp = (revLookup[b64.charCodeAt(i)] << 10) | (revLookup[b64.charCodeAt(i + 1)] << 4) | (revLookup[b64.charCodeAt(i + 2)] >> 2)
      arr[L++] = (tmp >> 8) & 0xFF
      arr[L++] = tmp & 0xFF
    }

    return arr
  }

  function tripletToBase64 (num) {
    return lookup[num >> 18 & 0x3F] + lookup[num >> 12 & 0x3F] + lookup[num >> 6 & 0x3F] + lookup[num & 0x3F]
  }

  function encodeChunk (uint8, start, end) {
    var tmp
    var output = []
```

```
    for (var i = start; i < end; i += 3) {
      tmp = (uint8[i] << 16) + (uint8[i + 1] << 8) + (uint8[i + 2])
      output.push(tripletToBase64(tmp))
    }
    return output.join('')
  }

  function fromByteArray (uint8) {
    var tmp
    var len = uint8.length
    var extraBytes = len % 3 // if we have 1 byte left, pad 2 bytes
    var output = ''
    var parts = []
    var maxChunkLength = 16383 // must be multiple of 3

    // go through the array every three bytes, we'll deal with trailing stuff later
    for (var i = 0, len2 = len - extraBytes; i < len2; i += maxChunkLength) {
      parts.push(encodeChunk(uint8, i, (i + maxChunkLength) > len2 ? len2 : (i +
maxChunkLength)))
    }

    // pad the end with zeros, but make sure to not forget the extra bytes
    if (extraBytes === 1) {
      tmp = uint8[len - 1]
      output += lookup[tmp >> 2]
```

```
    output += lookup[(tmp << 4) & 0x3F]
    output += '=='
  } else if (extraBytes === 2) {
    tmp = (uint8[len - 2] << 8) + (uint8[len - 1])
    output += lookup[tmp >> 10]
    output += lookup[(tmp >> 4) & 0x3F]
    output += lookup[(tmp << 2) & 0x3F]
    output += '='
  }

  parts.push(output)

  return parts.join('')
}

},{}],4:[function(require,module,exports){
(function (module, exports) {
  'use strict';

  // Utils
  function assert (val, msg) {
    if (!val) throw new Error(msg || 'Assertion failed');
  }
```

```
// Could use `inherits` module, but don't want to move from single file
// architecture yet.
function inherits (ctor, superCtor) {
  ctor.super_ = superCtor;
  var TempCtor = function () {};
  TempCtor.prototype = superCtor.prototype;
  ctor.prototype = new TempCtor();
  ctor.prototype.constructor = ctor;
}

// BN

function BN (number, base, endian) {
  if (BN.isBN(number)) {
    return number;
  }

  this.negative = 0;
  this.words = null;
  this.length = 0;

  // Reduction context
  this.red = null;
```

```
  if (number !== null) {
    if (base === 'le' || base === 'be') {
      endian = base;
      base = 10;
    }

    this._init(number || 0, base || 10, endian || 'be');
  }
}
if (typeof module === 'object') {
  module.exports = BN;
} else {
  exports.BN = BN;
}

BN.BN = BN;
BN.wordSize = 26;

var Buffer;
try {
  Buffer = require('buf' + 'fer').Buffer;
} catch (e) {
}
```

```
BN.isBN = function isBN (num) {
  if (num instanceof BN) {
    return true;
  }

  return num !== null && typeof num === 'object' &&
    num.constructor.wordSize === BN.wordSize && Array.isArray(num.words);
};

BN.max = function max (left, right) {
  if (left.cmp(right) > 0) return left;
  return right;
};

BN.min = function min (left, right) {
  if (left.cmp(right) < 0) return left;
  return right;
};

BN.prototype._init = function init (number, base, endian) {
  if (typeof number === 'number') {
    return this._initNumber(number, base, endian);
  }
```

```
if (typeof number === 'object') {
  return this._initArray(number, base, endian);
}

if (base === 'hex') {
  base = 16;
}
assert(base === (base | 0) && base >= 2 && base <= 36);

number = number.toString().replace(/\s+/g, '');
var start = 0;
if (number[0] === '-') {
  start++;
}

if (base === 16) {
  this._parseHex(number, start);
} else {
  this._parseBase(number, base, start);
}

if (number[0] === '-') {
  this.negative = 1;
}
```

```
  this.strip();

  if (endian !== 'le') return;

  this._initArray(this.toArray(), base, endian);
};

BN.prototype._initNumber = function _initNumber (number, base, endian) {
  if (number < 0) {
    this.negative = 1;
    number = -number;
  }
  if (number < 0x4000000) {
    this.words = [ number & 0x3ffffff ];
    this.length = 1;
  } else if (number < 0x10000000000000) {
    this.words = [
      number & 0x3ffffff,
      (number / 0x4000000) & 0x3ffffff
    ];
    this.length = 2;
  } else {
    assert(number < 0x20000000000000); // 2 ^ 53 (unsafe)
```

```
    this.words = [
      number & 0x3ffffff,
      (number / 0x4000000) & 0x3ffffff,
      1
    ];
    this.length = 3;
  }

  if (endian !== 'le') return;

  // Reverse the bytes
  this._initArray(this.toArray(), base, endian);
};

BN.prototype._initArray = function _initArray (number, base, endian) {
  // Perhaps a Uint8Array
  assert(typeof number.length === 'number');
  if (number.length <= 0) {
    this.words = [ 0 ];
    this.length = 1;
    return this;
  }

  this.length = Math.ceil(number.length / 3);
```

```
this.words = new Array(this.length);
for (var i = 0; i < this.length; i++) {
  this.words[i] = 0;
}

var j, w;
var off = 0;
if (endian === 'be') {
  for (i = number.length - 1, j = 0; i >= 0; i -= 3) {
    w = number[i] | (number[i - 1] << 8) | (number[i - 2] << 16);
    this.words[j] |= (w << off) & 0x3ffffff;
    this.words[j + 1] = (w >>> (26 - off)) & 0x3ffffff;
    off += 24;
    if (off >= 26) {
      off -= 26;
      j++;
    }
  }
} else if (endian === 'le') {
  for (i = 0, j = 0; i < number.length; i += 3) {
    w = number[i] | (number[i + 1] << 8) | (number[i + 2] << 16);
    this.words[j] |= (w << off) & 0x3ffffff;
    this.words[j + 1] = (w >>> (26 - off)) & 0x3ffffff;
    off += 24;
```

```
      if (off >= 26) {
        off -= 26;
        j++;
      }
    }
  }
  return this.strip();
};

function parseHex (str, start, end) {
  var r = 0;
  var len = Math.min(str.length, end);
  for (var i = start; i < len; i++) {
    var c = str.charCodeAt(i) - 48;

    r <<= 4;

    // 'a' - 'f'
    if (c >= 49 && c <= 54) {
      r |= c - 49 + 0xa;

    // 'A' - 'F'
    } else if (c >= 17 && c <= 22) {
      r |= c - 17 + 0xa;
```

```
    // '0' - '9'
    } else {
      r |= c & 0xf;
    }
  }
  return r;
}

BN.prototype._parseHex = function _parseHex (number, start) {
  // Create possibly bigger array to ensure that it fits the number
  this.length = Math.ceil((number.length - start) / 6);
  this.words = new Array(this.length);
  for (var i = 0; i < this.length; i++) {
    this.words[i] = 0;
  }

  var j, w;
  // Scan 24-bit chunks and add them to the number
  var off = 0;
  for (i = number.length - 6, j = 0; i >= start; i -= 6) {
    w = parseHex(number, i, i + 6);
    this.words[j] |= (w << off) & 0x3ffffff;
    // NOTE: `0x3fffff` is intentional here, 26bits max shift + 24bit hex limb
```

```
      this.words[j + 1] |= w >>> (26 - off) & 0x3fffff;
      off += 24;
      if (off >= 26) {
        off -= 26;
        j++;
      }
    }
    if (i + 6 !== start) {
      w = parseHex(number, start, i + 6);
      this.words[j] |= (w << off) & 0x3ffffff;
      this.words[j + 1] |= w >>> (26 - off) & 0x3fffff;
    }
    this.strip();
  };

  function parseBase (str, start, end, mul) {
    var r = 0;
    var len = Math.min(str.length, end);
    for (var i = start; i < len; i++) {
      var c = str.charCodeAt(i) - 48;

      r *= mul;

      // 'a'
```

```
    if (c >= 49) {
      r += c - 49 + 0xa;

    // 'A'
    } else if (c >= 17) {
      r += c - 17 + 0xa;

    // '0' - '9'
    } else {
      r += c;
    }
  }
  return r;
}

BN.prototype._parseBase = function _parseBase (number, base, start) {
  // Initialize as zero
  this.words = [ 0 ];
  this.length = 1;

  // Find length of limb in base
  for (var limbLen = 0, limbPow = 1; limbPow <= 0x3ffffff; limbPow *= base) {
    limbLen++;
  }
```

```
limbLen--;
limbPow = (limbPow / base) | 0;

var total = number.length - start;
var mod = total % limbLen;
var end = Math.min(total, total - mod) + start;

var word = 0;
for (var i = start; i < end; i += limbLen) {
  word = parseBase(number, i, i + limbLen, base);

  this.imuln(limbPow);
  if (this.words[0] + word < 0x4000000) {
    this.words[0] += word;
  } else {
    this._iaddn(word);
  }
}

if (mod !== 0) {
  var pow = 1;
  word = parseBase(number, i, number.length, base);

  for (i = 0; i < mod; i++) {
```

```
      pow *= base;
    }

    this.imuln(pow);
    if (this.words[0] + word < 0x4000000) {
      this.words[0] += word;
    } else {
      this._iaddn(word);
    }
  }
};

BN.prototype.copy = function copy (dest) {
  dest.words = new Array(this.length);
  for (var i = 0; i < this.length; i++) {
    dest.words[i] = this.words[i];
  }
  dest.length = this.length;
  dest.negative = this.negative;
  dest.red = this.red;
};

BN.prototype.clone = function clone () {
  var r = new BN(null);
```

```
  this.copy(r);
  return r;
};

BN.prototype._expand = function _expand (size) {
  while (this.length < size) {
    this.words[this.length++] = 0;
  }
  return this;
};

// Remove leading `0` from `this`
BN.prototype.strip = function strip () {
  while (this.length > 1 && this.words[this.length - 1] === 0) {
    this.length--;
  }
  return this._normSign();
};

BN.prototype._normSign = function _normSign () {
  // -0 = 0
  if (this.length === 1 && this.words[0] === 0) {
    this.negative = 0;
  }
```

```
  return this;
};

BN.prototype.inspect = function inspect () {
  return (this.red ? '<BN-R: ' : '<BN: ') + this.toString(16) + '>';
};

/*

var zeros = [];
var groupSizes = [];
var groupBases = [];

var s = '';
var i = -1;
while (++i < BN.wordSize) {
  zeros[i] = s;
  s += '0';
}
groupSizes[0] = 0;
groupSizes[1] = 0;
groupBases[0] = 0;
groupBases[1] = 0;
var base = 2 - 1;
```

```
while (++base < 36 + 1) {
  var groupSize = 0;
  var groupBase = 1;
  while (groupBase < (1 << BN.wordSize) / base) {
    groupBase *= base;
    groupSize += 1;
  }
  groupSizes[base] = groupSize;
  groupBases[base] = groupBase;
}

*/

var zeros = [
  '',
  '0',
  '00',
  '000',
  '0000',
  '00000',
  '000000',
  '0000000',
  '00000000',
  '000000000',
```

```
  '0000000000',
  '00000000000',
  '000000000000',
  '0000000000000',
  '00000000000000',
  '000000000000000',
  '0000000000000000',
  '00000000000000000',
  '000000000000000000',
  '0000000000000000000',
  '00000000000000000000',
  '000000000000000000000',
  '0000000000000000000000',
  '00000000000000000000000',
  '000000000000000000000000',
  '0000000000000000000000000'
];

var groupSizes = [
  0, 0,
  25, 16, 12, 11, 10, 9, 8,
  8, 7, 7, 7, 7, 6, 6,
  6, 6, 6, 6, 6, 5, 5,
  5, 5, 5, 5, 5, 5, 5,
```

```
  5, 5, 5, 5, 5, 5, 5
];

var groupBases = [
  0, 0,
  33554432, 43046721, 16777216, 48828125, 60466176, 40353607, 16777216,
  43046721, 10000000, 19487171, 35831808, 62748517, 7529536, 11390625,
  16777216, 24137569, 34012224, 47045881, 64000000, 4084101, 5153632,
  6436343, 7962624, 9765625, 11881376, 14348907, 17210368, 20511149,
  24300000, 28629151, 33554432, 39135393, 45435424, 52521875, 60466176
];

BN.prototype.toString = function toString (base, padding) {
  base = base || 10;
  padding = padding | 0 || 1;

  var out;
  if (base === 16 || base === 'hex') {
    out = '';
    var off = 0;
    var carry = 0;
    for (var i = 0; i < this.length; i++) {
      var w = this.words[i];
      var word = (((w << off) | carry) & 0xffffff).toString(16);
```

```
    carry = (w >>> (24 - off)) & 0xffffff;
    if (carry !== 0 || i !== this.length - 1) {
      out = zeros[6 - word.length] + word + out;
    } else {
      out = word + out;
    }
    off += 2;
    if (off >= 26) {
      off -= 26;
      i--;
    }
  }
  if (carry !== 0) {
    out = carry.toString(16) + out;
  }
  while (out.length % padding !== 0) {
    out = '0' + out;
  }
  if (this.negative !== 0) {
    out = '-' + out;
  }
  return out;
}
```

```
if (base === (base | 0) && base >= 2 && base <= 36) {
  // var groupSize = Math.floor(BN.wordSize * Math.LN2 / Math.log(base));
  var groupSize = groupSizes[base];
  // var groupBase = Math.pow(base, groupSize);
  var groupBase = groupBases[base];
  out = '';
  var c = this.clone();
  c.negative = 0;
  while (!c.isZero()) {
    var r = c.modn(groupBase).toString(base);
    c = c.idivn(groupBase);

    if (!c.isZero()) {
      out = zeros[groupSize - r.length] + r + out;
    } else {
      out = r + out;
    }
  }
  if (this.isZero()) {
    out = '0' + out;
  }
  while (out.length % padding !== 0) {
    out = '0' + out;
  }
```

```
    if (this.negative !== 0) {
      out = '-' + out;
    }
    return out;
  }

  assert(false, 'Base should be between 2 and 36');
};

BN.prototype.toNumber = function toNumber () {
  var ret = this.words[0];
  if (this.length === 2) {
    ret += this.words[1] * 0x4000000;
  } else if (this.length === 3 && this.words[2] === 0x01) {
    // NOTE: at this stage it is known that the top bit is set
    ret += 0x10000000000000 + (this.words[1] * 0x4000000);
  } else if (this.length > 2) {
    assert(false, 'Number can only safely store up to 53 bits');
  }
  return (this.negative !== 0) ? -ret : ret;
};

BN.prototype.toJSON = function toJSON () {
  return this.toString(16);
```

```
};

BN.prototype.toBuffer = function toBuffer (endian, length) {
  assert(typeof Buffer !== 'undefined');
  return this.toArrayLike(Buffer, endian, length);
};

BN.prototype.toArray = function toArray (endian, length) {
  return this.toArrayLike(Array, endian, length);
};

BN.prototype.toArrayLike = function toArrayLike (ArrayType, endian, length) {
  var byteLength = this.byteLength();
  var reqLength = length || Math.max(1, byteLength);
  assert(byteLength <= reqLength, 'byte array longer than desired length');
  assert(reqLength > 0, 'Requested array length <= 0');

  this.strip();
  var littleEndian = endian === 'le';
  var res = new ArrayType(reqLength);

  var b, i;
  var q = this.clone();
  if (!littleEndian) {
```

```
    // Assume big-endian
    for (i = 0; i < reqLength - byteLength; i++) {
      res[i] = 0;
    }

    for (i = 0; !q.isZero(); i++) {
      b = q.andln(0xff);
      q.iushrn(8);

      res[reqLength - i - 1] = b;
    }
  } else {
    for (i = 0; !q.isZero(); i++) {
      b = q.andln(0xff);
      q.iushrn(8);

      res[i] = b;
    }

    for (; i < reqLength; i++) {
      res[i] = 0;
    }
  }
```

```
  return res;
};

if (Math.clz32) {
  BN.prototype._countBits = function _countBits (w) {
    return 32 - Math.clz32(w);
  };
} else {
  BN.prototype._countBits = function _countBits (w) {
    var t = w;
    var r = 0;
    if (t >= 0x1000) {
      r += 13;
      t >>>= 13;
    }
    if (t >= 0x40) {
      r += 7;
      t >>>= 7;
    }
    if (t >= 0x8) {
      r += 4;
      t >>>= 4;
    }
    if (t >= 0x02) {
```

```
      r += 2;
      t >>>= 2;
    }
    return r + t;
  };
}

BN.prototype._zeroBits = function _zeroBits (w) {
  // Short-cut
  if (w === 0) return 26;

  var t = w;
  var r = 0;
  if ((t & 0x1fff) === 0) {
    r += 13;
    t >>>= 13;
  }
  if ((t & 0x7f) === 0) {
    r += 7;
    t >>>= 7;
  }
  if ((t & 0xf) === 0) {
    r += 4;
    t >>>= 4;
```

```
  }
  if ((t & 0x3) === 0) {
    r += 2;
    t >>>= 2;
  }
  if ((t & 0x1) === 0) {
    r++;
  }
  return r;
};

// Return number of used bits in a BN
BN.prototype.bitLength = function bitLength () {
  var w = this.words[this.length - 1];
  var hi = this._countBits(w);
  return (this.length - 1) * 26 + hi;
};

function toBitArray (num) {
  var w = new Array(num.bitLength());

  for (var bit = 0; bit < w.length; bit++) {
    var off = (bit / 26) | 0;
    var wbit = bit % 26;
```

```
    w[bit] = (num.words[off] & (1 << wbit)) >>> wbit;
  }

  return w;
}

// Number of trailing zero bits
BN.prototype.zeroBits = function zeroBits () {
  if (this.isZero()) return 0;

  var r = 0;
  for (var i = 0; i < this.length; i++) {
    var b = this._zeroBits(this.words[i]);
    r += b;
    if (b !== 26) break;
  }
  return r;
};

BN.prototype.byteLength = function byteLength () {
  return Math.ceil(this.bitLength() / 8);
};
```

```
BN.prototype.toTwos = function toTwos (width) {
  if (this.negative !== 0) {
    return this.abs().inotn(width).iaddn(1);
  }
  return this.clone();
};

BN.prototype.fromTwos = function fromTwos (width) {
  if (this.testn(width - 1)) {
    return this.notn(width).iaddn(1).ineg();
  }
  return this.clone();
};

BN.prototype.isNeg = function isNeg () {
  return this.negative !== 0;
};

// Return negative clone of `this`
BN.prototype.neg = function neg () {
  return this.clone().ineg();
};

BN.prototype.ineg = function ineg () {
```

```
  if (!this.isZero()) {
    this.negative ^= 1;
  }

  return this;
};

// Or `num` with `this` in-place
BN.prototype.iuor = function iuor (num) {
  while (this.length < num.length) {
    this.words[this.length++] = 0;
  }

  for (var i = 0; i < num.length; i++) {
    this.words[i] = this.words[i] | num.words[i];
  }

  return this.strip();
};

BN.prototype.ior = function ior (num) {
  assert((this.negative | num.negative) === 0);
  return this.iuor(num);
};
```

```
// Or `num` with `this`
BN.prototype.or = function or (num) {
  if (this.length > num.length) return this.clone().ior(num);
  return num.clone().ior(this);
};

BN.prototype.uor = function uor (num) {
  if (this.length > num.length) return this.clone().iuor(num);
  return num.clone().iuor(this);
};

// And `num` with `this` in-place
BN.prototype.iuand = function iuand (num) {
  // b = min-length(num, this)
  var b;
  if (this.length > num.length) {
    b = num;
  } else {
    b = this;
  }

  for (var i = 0; i < b.length; i++) {
    this.words[i] = this.words[i] & num.words[i];
```

```
  }

  this.length = b.length;

  return this.strip();
};

BN.prototype.iand = function iand (num) {
  assert((this.negative | num.negative) === 0);
  return this.iuand(num);
};

// And `num` with `this`
BN.prototype.and = function and (num) {
  if (this.length > num.length) return this.clone().iand(num);
  return num.clone().iand(this);
};

BN.prototype.uand = function uand (num) {
  if (this.length > num.length) return this.clone().iuand(num);
  return num.clone().iuand(this);
};

// Xor `num` with `this` in-place
```

```
BN.prototype.iuxor = function iuxor (num) {
 // a.length > b.length
 var a;
 var b;
 if (this.length > num.length) {
  a = this;
  b = num;
 } else {
  a = num;
  b = this;
 }

 for (var i = 0; i < b.length; i++) {
  this.words[i] = a.words[i] ^ b.words[i];
 }

 if (this !== a) {
  for (; i < a.length; i++) {
   this.words[i] = a.words[i];
  }
 }

 this.length = a.length;
```

```
  return this.strip();
};

BN.prototype.ixor = function ixor (num) {
  assert((this.negative | num.negative) === 0);
  return this.iuxor(num);
};

// Xor `num` with `this`
BN.prototype.xor = function xor (num) {
  if (this.length > num.length) return this.clone().ixor(num);
  return num.clone().ixor(this);
};

BN.prototype.uxor = function uxor (num) {
  if (this.length > num.length) return this.clone().iuxor(num);
  return num.clone().iuxor(this);
};

// Not ``this`` with ``width`` bitwidth
BN.prototype.inotn = function inotn (width) {
  assert(typeof width === 'number' && width >= 0);

  var bytesNeeded = Math.ceil(width / 26) | 0;
```

```
  var bitsLeft = width % 26;

  // Extend the buffer with leading zeroes
  this._expand(bytesNeeded);

  if (bitsLeft > 0) {
    bytesNeeded--;
  }

  // Handle complete words
  for (var i = 0; i < bytesNeeded; i++) {
    this.words[i] = ~this.words[i] & 0x3ffffff;
  }

  // Handle the residue
  if (bitsLeft > 0) {
    this.words[i] = ~this.words[i] & (0x3ffffff >> (26 - bitsLeft));
  }

  // And remove leading zeroes
  return this.strip();
};

BN.prototype.notn = function notn (width) {
```

```
  return this.clone().inotn(width);
};

// Set `bit` of `this`
BN.prototype.setn = function setn (bit, val) {
  assert(typeof bit === 'number' && bit >= 0);

  var off = (bit / 26) | 0;
  var wbit = bit % 26;

  this._expand(off + 1);

  if (val) {
    this.words[off] = this.words[off] | (1 << wbit);
  } else {
    this.words[off] = this.words[off] & ~(1 << wbit);
  }

  return this.strip();
};

// Add `num` to `this` in-place
BN.prototype.iadd = function iadd (num) {
  var r;
```

```
// negative + positive
if (this.negative !== 0 && num.negative === 0) {
  this.negative = 0;
  r = this.isub(num);
  this.negative ^= 1;
  return this._normSign();

// positive + negative
} else if (this.negative === 0 && num.negative !== 0) {
  num.negative = 0;
  r = this.isub(num);
  num.negative = 1;
  return r._normSign();
}

// a.length > b.length
var a, b;
if (this.length > num.length) {
  a = this;
  b = num;
} else {
  a = num;
  b = this;
```

```
}

var carry = 0;
for (var i = 0; i < b.length; i++) {
  r = (a.words[i] | 0) + (b.words[i] | 0) + carry;
  this.words[i] = r & 0x3ffffff;
  carry = r >>> 26;
}
for (; carry !== 0 && i < a.length; i++) {
  r = (a.words[i] | 0) + carry;
  this.words[i] = r & 0x3ffffff;
  carry = r >>> 26;
}

this.length = a.length;
if (carry !== 0) {
  this.words[this.length] = carry;
  this.length++;
// Copy the rest of the words
} else if (a !== this) {
  for (; i < a.length; i++) {
    this.words[i] = a.words[i];
  }
}
```

```
  return this;
};

// Add `num` to `this`
BN.prototype.add = function add (num) {
  var res;
  if (num.negative !== 0 && this.negative === 0) {
    num.negative = 0;
    res = this.sub(num);
    num.negative ^= 1;
    return res;
  } else if (num.negative === 0 && this.negative !== 0) {
    this.negative = 0;
    res = num.sub(this);
    this.negative = 1;
    return res;
  }

  if (this.length > num.length) return this.clone().iadd(num);

  return num.clone().iadd(this);
};
```

```
// Subtract `num` from `this` in-place
BN.prototype.isub = function isub (num) {
  // this - (-num) = this + num
  if (num.negative !== 0) {
    num.negative = 0;
    var r = this.iadd(num);
    num.negative = 1;
    return r._normSign();

  // -this - num = -(this + num)
  } else if (this.negative !== 0) {
    this.negative = 0;
    this.iadd(num);
    this.negative = 1;
    return this._normSign();
  }

  // At this point both numbers are positive
  var cmp = this.cmp(num);

  // Optimization - zeroify
  if (cmp === 0) {
    this.negative = 0;
    this.length = 1;
```

```
  this.words[0] = 0;
  return this;
}

// a > b
var a, b;
if (cmp > 0) {
  a = this;
  b = num;
} else {
  a = num;
  b = this;
}

var carry = 0;
for (var i = 0; i < b.length; i++) {
  r = (a.words[i] | 0) - (b.words[i] | 0) + carry;
  carry = r >> 26;
  this.words[i] = r & 0x3ffffff;
}
for (; carry !== 0 && i < a.length; i++) {
  r = (a.words[i] | 0) + carry;
  carry = r >> 26;
  this.words[i] = r & 0x3ffffff;
```

```
  }

  // Copy rest of the words
  if (carry === 0 && i < a.length && a !== this) {
    for (; i < a.length; i++) {
      this.words[i] = a.words[i];
    }
  }

  this.length = Math.max(this.length, i);

  if (a !== this) {
    this.negative = 1;
  }

  return this.strip();
};

// Subtract `num` from `this`
BN.prototype.sub = function sub (num) {
  return this.clone().isub(num);
};

function smallMulTo (self, num, out) {
```

```
out.negative = num.negative ^ self.negative;
var len = (self.length + num.length) | 0;
out.length = len;
len = (len - 1) | 0;

// Peel one iteration (compiler can't do it, because of code complexity)
var a = self.words[0] | 0;
var b = num.words[0] | 0;
var r = a * b;

var lo = r & 0x3ffffff;
var carry = (r / 0x4000000) | 0;
out.words[0] = lo;

for (var k = 1; k < len; k++) {
  // Sum all words with the same `i + j = k` and accumulate `ncarry`,
  // note that ncarry could be >= 0x3ffffff
  var ncarry = carry >>> 26;
  var rword = carry & 0x3ffffff;
  var maxJ = Math.min(k, num.length - 1);
  for (var j = Math.max(0, k - self.length + 1); j <= maxJ; j++) {
    var i = (k - j) | 0;
    a = self.words[i] | 0;
    b = num.words[j] | 0;
```

```
      r = a * b + rword;
      ncarry += (r / 0x4000000) | 0;
      rword = r & 0x3ffffff;
    }
    out.words[k] = rword | 0;
    carry = ncarry | 0;
  }
  if (carry !== 0) {
    out.words[k] = carry | 0;
  } else {
    out.length--;
  }

  return out.strip();
}

// TODO(indutny): it may be reasonable to omit it for users who don't need
// to work with 256-bit numbers, otherwise it gives 20% improvement for 256-
bit
// multiplication (like elliptic secp256k1).
var comb10MulTo = function comb10MulTo (self, num, out) {
  var a = self.words;
  var b = num.words;
  var o = out.words;
  var c = 0;
```

```
var lo;
var mid;
var hi;
var a0 = a[0] | 0;
var al0 = a0 & 0x1fff;
var ah0 = a0 >>> 13;
var a1 = a[1] | 0;
var al1 = a1 & 0x1fff;
var ah1 = a1 >>> 13;
var a2 = a[2] | 0;
var al2 = a2 & 0x1fff;
var ah2 = a2 >>> 13;
var a3 = a[3] | 0;
var al3 = a3 & 0x1fff;
var ah3 = a3 >>> 13;
var a4 = a[4] | 0;
var al4 = a4 & 0x1fff;
var ah4 = a4 >>> 13;
var a5 = a[5] | 0;
var al5 = a5 & 0x1fff;
var ah5 = a5 >>> 13;
var a6 = a[6] | 0;
var al6 = a6 & 0x1fff;
var ah6 = a6 >>> 13;
```

```
var a7 = a[7] | 0;
var al7 = a7 & 0x1fff;
var ah7 = a7 >>> 13;
var a8 = a[8] | 0;
var al8 = a8 & 0x1fff;
var ah8 = a8 >>> 13;
var a9 = a[9] | 0;
var al9 = a9 & 0x1fff;
var ah9 = a9 >>> 13;
var b0 = b[0] | 0;
var bl0 = b0 & 0x1fff;
var bh0 = b0 >>> 13;
var b1 = b[1] | 0;
var bl1 = b1 & 0x1fff;
var bh1 = b1 >>> 13;
var b2 = b[2] | 0;
var bl2 = b2 & 0x1fff;
var bh2 = b2 >>> 13;
var b3 = b[3] | 0;
var bl3 = b3 & 0x1fff;
var bh3 = b3 >>> 13;
var b4 = b[4] | 0;
var bl4 = b4 & 0x1fff;
var bh4 = b4 >>> 13;
```

```
var b5 = b[5] | 0;
var bl5 = b5 & 0x1fff;
var bh5 = b5 >>> 13;
var b6 = b[6] | 0;
var bl6 = b6 & 0x1fff;
var bh6 = b6 >>> 13;
var b7 = b[7] | 0;
var bl7 = b7 & 0x1fff;
var bh7 = b7 >>> 13;
var b8 = b[8] | 0;
var bl8 = b8 & 0x1fff;
var bh8 = b8 >>> 13;
var b9 = b[9] | 0;
var bl9 = b9 & 0x1fff;
var bh9 = b9 >>> 13;

out.negative = self.negative ^ num.negative;
out.length = 19;
/* k = 0 */
lo = Math.imul(al0, bl0);
mid = Math.imul(al0, bh0);
mid = (mid + Math.imul(ah0, bl0)) | 0;
hi = Math.imul(ah0, bh0);
var w0 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
```

```
c = (((hi + (mid >>> 13)) | 0) + (w0 >>> 26)) | 0;
w0 &= 0x3ffffff;
/* k = 1 */
lo = Math.imul(al1, bl0);
mid = Math.imul(al1, bh0);
mid = (mid + Math.imul(ah1, bl0)) | 0;
hi = Math.imul(ah1, bh0);
lo = (lo + Math.imul(al0, bl1)) | 0;
mid = (mid + Math.imul(al0, bh1)) | 0;
mid = (mid + Math.imul(ah0, bl1)) | 0;
hi = (hi + Math.imul(ah0, bh1)) | 0;
var w1 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w1 >>> 26)) | 0;
w1 &= 0x3ffffff;
/* k = 2 */
lo = Math.imul(al2, bl0);
mid = Math.imul(al2, bh0);
mid = (mid + Math.imul(ah2, bl0)) | 0;
hi = Math.imul(ah2, bh0);
lo = (lo + Math.imul(al1, bl1)) | 0;
mid = (mid + Math.imul(al1, bh1)) | 0;
mid = (mid + Math.imul(ah1, bl1)) | 0;
hi = (hi + Math.imul(ah1, bh1)) | 0;
lo = (lo + Math.imul(al0, bl2)) | 0;
```

```
mid = (mid + Math.imul(al0, bh2)) | 0;
mid = (mid + Math.imul(ah0, bl2)) | 0;
hi = (hi + Math.imul(ah0, bh2)) | 0;
var w2 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w2 >>> 26)) | 0;
w2 &= 0x3ffffff;
/* k = 3 */
lo = Math.imul(al3, bl0);
mid = Math.imul(al3, bh0);
mid = (mid + Math.imul(ah3, bl0)) | 0;
hi = Math.imul(ah3, bh0);
lo = (lo + Math.imul(al2, bl1)) | 0;
mid = (mid + Math.imul(al2, bh1)) | 0;
mid = (mid + Math.imul(ah2, bl1)) | 0;
hi = (hi + Math.imul(ah2, bh1)) | 0;
lo = (lo + Math.imul(al1, bl2)) | 0;
mid = (mid + Math.imul(al1, bh2)) | 0;
mid = (mid + Math.imul(ah1, bl2)) | 0;
hi = (hi + Math.imul(ah1, bh2)) | 0;
lo = (lo + Math.imul(al0, bl3)) | 0;
mid = (mid + Math.imul(al0, bh3)) | 0;
mid = (mid + Math.imul(ah0, bl3)) | 0;
hi = (hi + Math.imul(ah0, bh3)) | 0;
var w3 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
```

```
c = (((hi + (mid >>> 13)) | 0) + (w3 >>> 26)) | 0;
w3 &= 0x3ffffff;
/* k = 4 */
lo = Math.imul(al4, bl0);
mid = Math.imul(al4, bh0);
mid = (mid + Math.imul(ah4, bl0)) | 0;
hi = Math.imul(ah4, bh0);
lo = (lo + Math.imul(al3, bl1)) | 0;
mid = (mid + Math.imul(al3, bh1)) | 0;
mid = (mid + Math.imul(ah3, bl1)) | 0;
hi = (hi + Math.imul(ah3, bh1)) | 0;
lo = (lo + Math.imul(al2, bl2)) | 0;
mid = (mid + Math.imul(al2, bh2)) | 0;
mid = (mid + Math.imul(ah2, bl2)) | 0;
hi = (hi + Math.imul(ah2, bh2)) | 0;
lo = (lo + Math.imul(al1, bl3)) | 0;
mid = (mid + Math.imul(al1, bh3)) | 0;
mid = (mid + Math.imul(ah1, bl3)) | 0;
hi = (hi + Math.imul(ah1, bh3)) | 0;
lo = (lo + Math.imul(al0, bl4)) | 0;
mid = (mid + Math.imul(al0, bh4)) | 0;
mid = (mid + Math.imul(ah0, bl4)) | 0;
hi = (hi + Math.imul(ah0, bh4)) | 0;
var w4 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
```

```
c = (((hi + (mid >>> 13)) | 0) + (w4 >>> 26)) | 0;
w4 &= 0x3ffffff;
/* k = 5 */
lo = Math.imul(al5, bl0);
mid = Math.imul(al5, bh0);
mid = (mid + Math.imul(ah5, bl0)) | 0;
hi = Math.imul(ah5, bh0);
lo = (lo + Math.imul(al4, bl1)) | 0;
mid = (mid + Math.imul(al4, bh1)) | 0;
mid = (mid + Math.imul(ah4, bl1)) | 0;
hi = (hi + Math.imul(ah4, bh1)) | 0;
lo = (lo + Math.imul(al3, bl2)) | 0;
mid = (mid + Math.imul(al3, bh2)) | 0;
mid = (mid + Math.imul(ah3, bl2)) | 0;
hi = (hi + Math.imul(ah3, bh2)) | 0;
lo = (lo + Math.imul(al2, bl3)) | 0;
mid = (mid + Math.imul(al2, bh3)) | 0;
mid = (mid + Math.imul(ah2, bl3)) | 0;
hi = (hi + Math.imul(ah2, bh3)) | 0;
lo = (lo + Math.imul(al1, bl4)) | 0;
mid = (mid + Math.imul(al1, bh4)) | 0;
mid = (mid + Math.imul(ah1, bl4)) | 0;
hi = (hi + Math.imul(ah1, bh4)) | 0;
lo = (lo + Math.imul(al0, bl5)) | 0;
```

```
mid = (mid + Math.imul(al0, bh5)) | 0;
mid = (mid + Math.imul(ah0, bl5)) | 0;
hi = (hi + Math.imul(ah0, bh5)) | 0;
var w5 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w5 >>> 26)) | 0;
w5 &= 0x3ffffff;
/* k = 6 */
lo = Math.imul(al6, bl0);
mid = Math.imul(al6, bh0);
mid = (mid + Math.imul(ah6, bl0)) | 0;
hi = Math.imul(ah6, bh0);
lo = (lo + Math.imul(al5, bl1)) | 0;
mid = (mid + Math.imul(al5, bh1)) | 0;
mid = (mid + Math.imul(ah5, bl1)) | 0;
hi = (hi + Math.imul(ah5, bh1)) | 0;
lo = (lo + Math.imul(al4, bl2)) | 0;
mid = (mid + Math.imul(al4, bh2)) | 0;
mid = (mid + Math.imul(ah4, bl2)) | 0;
hi = (hi + Math.imul(ah4, bh2)) | 0;
lo = (lo + Math.imul(al3, bl3)) | 0;
mid = (mid + Math.imul(al3, bh3)) | 0;
mid = (mid + Math.imul(ah3, bl3)) | 0;
hi = (hi + Math.imul(ah3, bh3)) | 0;
lo = (lo + Math.imul(al2, bl4)) | 0;
```

```
mid = (mid + Math.imul(al2, bh4)) | 0;
mid = (mid + Math.imul(ah2, bl4)) | 0;
hi = (hi + Math.imul(ah2, bh4)) | 0;
lo = (lo + Math.imul(al1, bl5)) | 0;
mid = (mid + Math.imul(al1, bh5)) | 0;
mid = (mid + Math.imul(ah1, bl5)) | 0;
hi = (hi + Math.imul(ah1, bh5)) | 0;
lo = (lo + Math.imul(al0, bl6)) | 0;
mid = (mid + Math.imul(al0, bh6)) | 0;
mid = (mid + Math.imul(ah0, bl6)) | 0;
hi = (hi + Math.imul(ah0, bh6)) | 0;
var w6 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w6 >>> 26)) | 0;
w6 &= 0x3ffffff;
/* k = 7 */
lo = Math.imul(al7, bl0);
mid = Math.imul(al7, bh0);
mid = (mid + Math.imul(ah7, bl0)) | 0;
hi = Math.imul(ah7, bh0);
lo = (lo + Math.imul(al6, bl1)) | 0;
mid = (mid + Math.imul(al6, bh1)) | 0;
mid = (mid + Math.imul(ah6, bl1)) | 0;
hi = (hi + Math.imul(ah6, bh1)) | 0;
lo = (lo + Math.imul(al5, bl2)) | 0;
```

```
mid = (mid + Math.imul(al5, bh2)) | 0;
mid = (mid + Math.imul(ah5, bl2)) | 0;
hi = (hi + Math.imul(ah5, bh2)) | 0;
lo = (lo + Math.imul(al4, bl3)) | 0;
mid = (mid + Math.imul(al4, bh3)) | 0;
mid = (mid + Math.imul(ah4, bl3)) | 0;
hi = (hi + Math.imul(ah4, bh3)) | 0;
lo = (lo + Math.imul(al3, bl4)) | 0;
mid = (mid + Math.imul(al3, bh4)) | 0;
mid = (mid + Math.imul(ah3, bl4)) | 0;
hi = (hi + Math.imul(ah3, bh4)) | 0;
lo = (lo + Math.imul(al2, bl5)) | 0;
mid = (mid + Math.imul(al2, bh5)) | 0;
mid = (mid + Math.imul(ah2, bl5)) | 0;
hi = (hi + Math.imul(ah2, bh5)) | 0;
lo = (lo + Math.imul(al1, bl6)) | 0;
mid = (mid + Math.imul(al1, bh6)) | 0;
mid = (mid + Math.imul(ah1, bl6)) | 0;
hi = (hi + Math.imul(ah1, bh6)) | 0;
lo = (lo + Math.imul(al0, bl7)) | 0;
mid = (mid + Math.imul(al0, bh7)) | 0;
mid = (mid + Math.imul(ah0, bl7)) | 0;
hi = (hi + Math.imul(ah0, bh7)) | 0;
var w7 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
```

```
c = (((hi + (mid >>> 13)) | 0) + (w7 >>> 26)) | 0;
w7 &= 0x3ffffff;
/* k = 8 */
lo = Math.imul(al8, bl0);
mid = Math.imul(al8, bh0);
mid = (mid + Math.imul(ah8, bl0)) | 0;
hi = Math.imul(ah8, bh0);
lo = (lo + Math.imul(al7, bl1)) | 0;
mid = (mid + Math.imul(al7, bh1)) | 0;
mid = (mid + Math.imul(ah7, bl1)) | 0;
hi = (hi + Math.imul(ah7, bh1)) | 0;
lo = (lo + Math.imul(al6, bl2)) | 0;
mid = (mid + Math.imul(al6, bh2)) | 0;
mid = (mid + Math.imul(ah6, bl2)) | 0;
hi = (hi + Math.imul(ah6, bh2)) | 0;
lo = (lo + Math.imul(al5, bl3)) | 0;
mid = (mid + Math.imul(al5, bh3)) | 0;
mid = (mid + Math.imul(ah5, bl3)) | 0;
hi = (hi + Math.imul(ah5, bh3)) | 0;
lo = (lo + Math.imul(al4, bl4)) | 0;
mid = (mid + Math.imul(al4, bh4)) | 0;
mid = (mid + Math.imul(ah4, bl4)) | 0;
hi = (hi + Math.imul(ah4, bh4)) | 0;
lo = (lo + Math.imul(al3, bl5)) | 0;
```

```
mid = (mid + Math.imul(al3, bh5)) | 0;
mid = (mid + Math.imul(ah3, bl5)) | 0;
hi = (hi + Math.imul(ah3, bh5)) | 0;
lo = (lo + Math.imul(al2, bl6)) | 0;
mid = (mid + Math.imul(al2, bh6)) | 0;
mid = (mid + Math.imul(ah2, bl6)) | 0;
hi = (hi + Math.imul(ah2, bh6)) | 0;
lo = (lo + Math.imul(al1, bl7)) | 0;
mid = (mid + Math.imul(al1, bh7)) | 0;
mid = (mid + Math.imul(ah1, bl7)) | 0;
hi = (hi + Math.imul(ah1, bh7)) | 0;
lo = (lo + Math.imul(al0, bl8)) | 0;
mid = (mid + Math.imul(al0, bh8)) | 0;
mid = (mid + Math.imul(ah0, bl8)) | 0;
hi = (hi + Math.imul(ah0, bh8)) | 0;
var w8 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w8 >>> 26)) | 0;
w8 &= 0x3ffffff;
/* k = 9 */
lo = Math.imul(al9, bl0);
mid = Math.imul(al9, bh0);
mid = (mid + Math.imul(ah9, bl0)) | 0;
hi = Math.imul(ah9, bh0);
lo = (lo + Math.imul(al8, bl1)) | 0;
```

```
mid = (mid + Math.imul(al8, bh1)) | 0;
mid = (mid + Math.imul(ah8, bl1)) | 0;
hi = (hi + Math.imul(ah8, bh1)) | 0;
lo = (lo + Math.imul(al7, bl2)) | 0;
mid = (mid + Math.imul(al7, bh2)) | 0;
mid = (mid + Math.imul(ah7, bl2)) | 0;
hi = (hi + Math.imul(ah7, bh2)) | 0;
lo = (lo + Math.imul(al6, bl3)) | 0;
mid = (mid + Math.imul(al6, bh3)) | 0;
mid = (mid + Math.imul(ah6, bl3)) | 0;
hi = (hi + Math.imul(ah6, bh3)) | 0;
lo = (lo + Math.imul(al5, bl4)) | 0;
mid = (mid + Math.imul(al5, bh4)) | 0;
mid = (mid + Math.imul(ah5, bl4)) | 0;
hi = (hi + Math.imul(ah5, bh4)) | 0;
lo = (lo + Math.imul(al4, bl5)) | 0;
mid = (mid + Math.imul(al4, bh5)) | 0;
mid = (mid + Math.imul(ah4, bl5)) | 0;
hi = (hi + Math.imul(ah4, bh5)) | 0;
lo = (lo + Math.imul(al3, bl6)) | 0;
mid = (mid + Math.imul(al3, bh6)) | 0;
mid = (mid + Math.imul(ah3, bl6)) | 0;
hi = (hi + Math.imul(ah3, bh6)) | 0;
lo = (lo + Math.imul(al2, bl7)) | 0;
```

```
mid = (mid + Math.imul(al2, bh7)) | 0;
mid = (mid + Math.imul(ah2, bl7)) | 0;
hi = (hi + Math.imul(ah2, bh7)) | 0;
lo = (lo + Math.imul(al1, bl8)) | 0;
mid = (mid + Math.imul(al1, bh8)) | 0;
mid = (mid + Math.imul(ah1, bl8)) | 0;
hi = (hi + Math.imul(ah1, bh8)) | 0;
lo = (lo + Math.imul(al0, bl9)) | 0;
mid = (mid + Math.imul(al0, bh9)) | 0;
mid = (mid + Math.imul(ah0, bl9)) | 0;
hi = (hi + Math.imul(ah0, bh9)) | 0;
var w9 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w9 >>> 26)) | 0;
w9 &= 0x3ffffff;
/* k = 10 */
lo = Math.imul(al9, bl1);
mid = Math.imul(al9, bh1);
mid = (mid + Math.imul(ah9, bl1)) | 0;
hi = Math.imul(ah9, bh1);
lo = (lo + Math.imul(al8, bl2)) | 0;
mid = (mid + Math.imul(al8, bh2)) | 0;
mid = (mid + Math.imul(ah8, bl2)) | 0;
hi = (hi + Math.imul(ah8, bh2)) | 0;
lo = (lo + Math.imul(al7, bl3)) | 0;
```

```
mid = (mid + Math.imul(al7, bh3)) | 0;
mid = (mid + Math.imul(ah7, bl3)) | 0;
hi = (hi + Math.imul(ah7, bh3)) | 0;
lo = (lo + Math.imul(al6, bl4)) | 0;
mid = (mid + Math.imul(al6, bh4)) | 0;
mid = (mid + Math.imul(ah6, bl4)) | 0;
hi = (hi + Math.imul(ah6, bh4)) | 0;
lo = (lo + Math.imul(al5, bl5)) | 0;
mid = (mid + Math.imul(al5, bh5)) | 0;
mid = (mid + Math.imul(ah5, bl5)) | 0;
hi = (hi + Math.imul(ah5, bh5)) | 0;
lo = (lo + Math.imul(al4, bl6)) | 0;
mid = (mid + Math.imul(al4, bh6)) | 0;
mid = (mid + Math.imul(ah4, bl6)) | 0;
hi = (hi + Math.imul(ah4, bh6)) | 0;
lo = (lo + Math.imul(al3, bl7)) | 0;
mid = (mid + Math.imul(al3, bh7)) | 0;
mid = (mid + Math.imul(ah3, bl7)) | 0;
hi = (hi + Math.imul(ah3, bh7)) | 0;
lo = (lo + Math.imul(al2, bl8)) | 0;
mid = (mid + Math.imul(al2, bh8)) | 0;
mid = (mid + Math.imul(ah2, bl8)) | 0;
hi = (hi + Math.imul(ah2, bh8)) | 0;
lo = (lo + Math.imul(al1, bl9)) | 0;
```

```
mid = (mid + Math.imul(al1, bh9)) | 0;
mid = (mid + Math.imul(ah1, bl9)) | 0;
hi = (hi + Math.imul(ah1, bh9)) | 0;
var w10 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w10 >>> 26)) | 0;
w10 &= 0x3ffffff;
/* k = 11 */
lo = Math.imul(al9, bl2);
mid = Math.imul(al9, bh2);
mid = (mid + Math.imul(ah9, bl2)) | 0;
hi = Math.imul(ah9, bh2);
lo = (lo + Math.imul(al8, bl3)) | 0;
mid = (mid + Math.imul(al8, bh3)) | 0;
mid = (mid + Math.imul(ah8, bl3)) | 0;
hi = (hi + Math.imul(ah8, bh3)) | 0;
lo = (lo + Math.imul(al7, bl4)) | 0;
mid = (mid + Math.imul(al7, bh4)) | 0;
mid = (mid + Math.imul(ah7, bl4)) | 0;
hi = (hi + Math.imul(ah7, bh4)) | 0;
lo = (lo + Math.imul(al6, bl5)) | 0;
mid = (mid + Math.imul(al6, bh5)) | 0;
mid = (mid + Math.imul(ah6, bl5)) | 0;
hi = (hi + Math.imul(ah6, bh5)) | 0;
lo = (lo + Math.imul(al5, bl6)) | 0;
```

```
mid = (mid + Math.imul(al5, bh6)) | 0;
mid = (mid + Math.imul(ah5, bl6)) | 0;
hi = (hi + Math.imul(ah5, bh6)) | 0;
lo = (lo + Math.imul(al4, bl7)) | 0;
mid = (mid + Math.imul(al4, bh7)) | 0;
mid = (mid + Math.imul(ah4, bl7)) | 0;
hi = (hi + Math.imul(ah4, bh7)) | 0;
lo = (lo + Math.imul(al3, bl8)) | 0;
mid = (mid + Math.imul(al3, bh8)) | 0;
mid = (mid + Math.imul(ah3, bl8)) | 0;
hi = (hi + Math.imul(ah3, bh8)) | 0;
lo = (lo + Math.imul(al2, bl9)) | 0;
mid = (mid + Math.imul(al2, bh9)) | 0;
mid = (mid + Math.imul(ah2, bl9)) | 0;
hi = (hi + Math.imul(ah2, bh9)) | 0;
var w11 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w11 >>> 26)) | 0;
w11 &= 0x3ffffff;
/* k = 12 */
lo = Math.imul(al9, bl3);
mid = Math.imul(al9, bh3);
mid = (mid + Math.imul(ah9, bl3)) | 0;
hi = Math.imul(ah9, bh3);
lo = (lo + Math.imul(al8, bl4)) | 0;
```

```
mid = (mid + Math.imul(al8, bh4)) | 0;
mid = (mid + Math.imul(ah8, bl4)) | 0;
hi = (hi + Math.imul(ah8, bh4)) | 0;
lo = (lo + Math.imul(al7, bl5)) | 0;
mid = (mid + Math.imul(al7, bh5)) | 0;
mid = (mid + Math.imul(ah7, bl5)) | 0;
hi = (hi + Math.imul(ah7, bh5)) | 0;
lo = (lo + Math.imul(al6, bl6)) | 0;
mid = (mid + Math.imul(al6, bh6)) | 0;
mid = (mid + Math.imul(ah6, bl6)) | 0;
hi = (hi + Math.imul(ah6, bh6)) | 0;
lo = (lo + Math.imul(al5, bl7)) | 0;
mid = (mid + Math.imul(al5, bh7)) | 0;
mid = (mid + Math.imul(ah5, bl7)) | 0;
hi = (hi + Math.imul(ah5, bh7)) | 0;
lo = (lo + Math.imul(al4, bl8)) | 0;
mid = (mid + Math.imul(al4, bh8)) | 0;
mid = (mid + Math.imul(ah4, bl8)) | 0;
hi = (hi + Math.imul(ah4, bh8)) | 0;
lo = (lo + Math.imul(al3, bl9)) | 0;
mid = (mid + Math.imul(al3, bh9)) | 0;
mid = (mid + Math.imul(ah3, bl9)) | 0;
hi = (hi + Math.imul(ah3, bh9)) | 0;
var w12 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
```

```
c = (((hi + (mid >>> 13)) | 0) + (w12 >>> 26)) | 0;
w12 &= 0x3ffffff;
/* k = 13 */
lo = Math.imul(al9, bl4);
mid = Math.imul(al9, bh4);
mid = (mid + Math.imul(ah9, bl4)) | 0;
hi = Math.imul(ah9, bh4);
lo = (lo + Math.imul(al8, bl5)) | 0;
mid = (mid + Math.imul(al8, bh5)) | 0;
mid = (mid + Math.imul(ah8, bl5)) | 0;
hi = (hi + Math.imul(ah8, bh5)) | 0;
lo = (lo + Math.imul(al7, bl6)) | 0;
mid = (mid + Math.imul(al7, bh6)) | 0;
mid = (mid + Math.imul(ah7, bl6)) | 0;
hi = (hi + Math.imul(ah7, bh6)) | 0;
lo = (lo + Math.imul(al6, bl7)) | 0;
mid = (mid + Math.imul(al6, bh7)) | 0;
mid = (mid + Math.imul(ah6, bl7)) | 0;
hi = (hi + Math.imul(ah6, bh7)) | 0;
lo = (lo + Math.imul(al5, bl8)) | 0;
mid = (mid + Math.imul(al5, bh8)) | 0;
mid = (mid + Math.imul(ah5, bl8)) | 0;
hi = (hi + Math.imul(ah5, bh8)) | 0;
lo = (lo + Math.imul(al4, bl9)) | 0;
```

```
mid = (mid + Math.imul(al4, bh9)) | 0;
mid = (mid + Math.imul(ah4, bl9)) | 0;
hi = (hi + Math.imul(ah4, bh9)) | 0;
var w13 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w13 >>> 26)) | 0;
w13 &= 0x3ffffff;
/* k = 14 */
lo = Math.imul(al9, bl5);
mid = Math.imul(al9, bh5);
mid = (mid + Math.imul(ah9, bl5)) | 0;
hi = Math.imul(ah9, bh5);
lo = (lo + Math.imul(al8, bl6)) | 0;
mid = (mid + Math.imul(al8, bh6)) | 0;
mid = (mid + Math.imul(ah8, bl6)) | 0;
hi = (hi + Math.imul(ah8, bh6)) | 0;
lo = (lo + Math.imul(al7, bl7)) | 0;
mid = (mid + Math.imul(al7, bh7)) | 0;
mid = (mid + Math.imul(ah7, bl7)) | 0;
hi = (hi + Math.imul(ah7, bh7)) | 0;
lo = (lo + Math.imul(al6, bl8)) | 0;
mid = (mid + Math.imul(al6, bh8)) | 0;
mid = (mid + Math.imul(ah6, bl8)) | 0;
hi = (hi + Math.imul(ah6, bh8)) | 0;
lo = (lo + Math.imul(al5, bl9)) | 0;
```

```
mid = (mid + Math.imul(al5, bh9)) | 0;
mid = (mid + Math.imul(ah5, bl9)) | 0;
hi = (hi + Math.imul(ah5, bh9)) | 0;
var w14 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w14 >>> 26)) | 0;
w14 &= 0x3ffffff;
/* k = 15 */
lo = Math.imul(al9, bl6);
mid = Math.imul(al9, bh6);
mid = (mid + Math.imul(ah9, bl6)) | 0;
hi = Math.imul(ah9, bh6);
lo = (lo + Math.imul(al8, bl7)) | 0;
mid = (mid + Math.imul(al8, bh7)) | 0;
mid = (mid + Math.imul(ah8, bl7)) | 0;
hi = (hi + Math.imul(ah8, bh7)) | 0;
lo = (lo + Math.imul(al7, bl8)) | 0;
mid = (mid + Math.imul(al7, bh8)) | 0;
mid = (mid + Math.imul(ah7, bl8)) | 0;
hi = (hi + Math.imul(ah7, bh8)) | 0;
lo = (lo + Math.imul(al6, bl9)) | 0;
mid = (mid + Math.imul(al6, bh9)) | 0;
mid = (mid + Math.imul(ah6, bl9)) | 0;
hi = (hi + Math.imul(ah6, bh9)) | 0;
var w15 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
```

```
c = (((hi + (mid >>> 13)) | 0) + (w15 >>> 26)) | 0;
w15 &= 0x3ffffff;
/* k = 16 */
lo = Math.imul(al9, bl7);
mid = Math.imul(al9, bh7);
mid = (mid + Math.imul(ah9, bl7)) | 0;
hi = Math.imul(ah9, bh7);
lo = (lo + Math.imul(al8, bl8)) | 0;
mid = (mid + Math.imul(al8, bh8)) | 0;
mid = (mid + Math.imul(ah8, bl8)) | 0;
hi = (hi + Math.imul(ah8, bh8)) | 0;
lo = (lo + Math.imul(al7, bl9)) | 0;
mid = (mid + Math.imul(al7, bh9)) | 0;
mid = (mid + Math.imul(ah7, bl9)) | 0;
hi = (hi + Math.imul(ah7, bh9)) | 0;
var w16 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w16 >>> 26)) | 0;
w16 &= 0x3ffffff;
/* k = 17 */
lo = Math.imul(al9, bl8);
mid = Math.imul(al9, bh8);
mid = (mid + Math.imul(ah9, bl8)) | 0;
hi = Math.imul(ah9, bh8);
lo = (lo + Math.imul(al8, bl9)) | 0;
```

```
mid = (mid + Math.imul(al8, bh9)) | 0;
mid = (mid + Math.imul(ah8, bl9)) | 0;
hi = (hi + Math.imul(ah8, bh9)) | 0;
var w17 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w17 >>> 26)) | 0;
w17 &= 0x3ffffff;
/* k = 18 */
lo = Math.imul(al9, bl9);
mid = Math.imul(al9, bh9);
mid = (mid + Math.imul(ah9, bl9)) | 0;
hi = Math.imul(ah9, bh9);
var w18 = (((c + lo) | 0) + ((mid & 0x1fff) << 13)) | 0;
c = (((hi + (mid >>> 13)) | 0) + (w18 >>> 26)) | 0;
w18 &= 0x3ffffff;
o[0] = w0;
o[1] = w1;
o[2] = w2;
o[3] = w3;
o[4] = w4;
o[5] = w5;
o[6] = w6;
o[7] = w7;
o[8] = w8;
o[9] = w9;
```

```
  o[10] = w10;
  o[11] = w11;
  o[12] = w12;
  o[13] = w13;
  o[14] = w14;
  o[15] = w15;
  o[16] = w16;
  o[17] = w17;
  o[18] = w18;
  if (c !== 0) {
    o[19] = c;
    out.length++;
  }
  return out;
};

// Polyfill comb
if (!Math.imul) {
  comb10MulTo = smallMulTo;
}

function bigMulTo (self, num, out) {
  out.negative = num.negative ^ self.negative;
  out.length = self.length + num.length;
```

```
var carry = 0;
var hncarry = 0;
for (var k = 0; k < out.length - 1; k++) {
  // Sum all words with the same `i + j = k` and accumulate `ncarry`,
  // note that ncarry could be >= 0x3ffffff
  var ncarry = hncarry;
  hncarry = 0;
  var rword = carry & 0x3ffffff;
  var maxJ = Math.min(k, num.length - 1);
  for (var j = Math.max(0, k - self.length + 1); j <= maxJ; j++) {
    var i = k - j;
    var a = self.words[i] | 0;
    var b = num.words[j] | 0;
    var r = a * b;

    var lo = r & 0x3ffffff;
    ncarry = (ncarry + ((r / 0x4000000) | 0)) | 0;
    lo = (lo + rword) | 0;
    rword = lo & 0x3ffffff;
    ncarry = (ncarry + (lo >>> 26)) | 0;

    hncarry += ncarry >>> 26;
    ncarry &= 0x3ffffff;
```

```
    }
    out.words[k] = rword;
    carry = ncarry;
    ncarry = hncarry;
  }
  if (carry !== 0) {
    out.words[k] = carry;
  } else {
    out.length--;
  }

  return out.strip();
}

function jumboMulTo (self, num, out) {
  var fftm = new FFTM();
  return fftm.mulp(self, num, out);
}

BN.prototype.mulTo = function mulTo (num, out) {
  var res;
  var len = this.length + num.length;
  if (this.length === 10 && num.length === 10) {
    res = comb10MulTo(this, num, out);
```

```
  } else if (len < 63) {
    res = smallMulTo(this, num, out);
  } else if (len < 1024) {
    res = bigMulTo(this, num, out);
  } else {
    res = jumboMulTo(this, num, out);
  }

  return res;
};

// Cooley-Tukey algorithm for FFT
// slightly revisited to rely on looping instead of recursion

function FFTM (x, y) {
  this.x = x;
  this.y = y;
}

FFTM.prototype.makeRBT = function makeRBT (N) {
  var t = new Array(N);
  var l = BN.prototype._countBits(N) - 1;
  for (var i = 0; i < N; i++) {
    t[i] = this.revBin(i, l, N);
```

```
  }

  return t;
};

// Returns binary-reversed representation of `x`
FFTM.prototype.revBin = function revBin (x, l, N) {
  if (x === 0 || x === N - 1) return x;

  var rb = 0;
  for (var i = 0; i < l; i++) {
    rb |= (x & 1) << (l - i - 1);
    x >>= 1;
  }

  return rb;
};

// Performs "tweedling" phase, therefore 'emulating'
// behaviour of the recursive algorithm
FFTM.prototype.permute = function permute (rbt, rws, iws, rtws, itws, N) {
  for (var i = 0; i < N; i++) {
    rtws[i] = rws[rbt[i]];
    itws[i] = iws[rbt[i]];
```

```
  }
};

FFTM.prototype.transform = function transform (rws, iws, rtws, itws, N, rbt) {
  this.permute(rbt, rws, iws, rtws, itws, N);

  for (var s = 1; s < N; s <<= 1) {
    var l = s << 1;

    var rtwdf = Math.cos(2 * Math.PI / l);
    var itwdf = Math.sin(2 * Math.PI / l);

    for (var p = 0; p < N; p += l) {
      var rtwdf_ = rtwdf;
      var itwdf_ = itwdf;

      for (var j = 0; j < s; j++) {
        var re = rtws[p + j];
        var ie = itws[p + j];

        var ro = rtws[p + j + s];
        var io = itws[p + j + s];

        var rx = rtwdf_ * ro - itwdf_ * io;
```

```
        io = rtwdf_ * io + itwdf_ * ro;
        ro = rx;

        rtws[p + j] = re + ro;
        itws[p + j] = ie + io;

        rtws[p + j + s] = re - ro;
        itws[p + j + s] = ie - io;

        /* jshint maxdepth : false */
        if (j !== l) {
          rx = rtwdf * rtwdf_ - itwdf * itwdf_;

          itwdf_ = rtwdf * itwdf_ + itwdf * rtwdf_;
          rtwdf_ = rx;
        }
      }
    }
  }
};

FFTM.prototype.guessLen13b = function guessLen13b (n, m) {
  var N = Math.max(m, n) | 1;
```

```
  var odd = N & 1;
  var i = 0;
  for (N = N / 2 | 0; N; N = N >>> 1) {
    i++;
  }

  return 1 << i + 1 + odd;
};

FFTM.prototype.conjugate = function conjugate (rws, iws, N) {
  if (N <= 1) return;

  for (var i = 0; i < N / 2; i++) {
    var t = rws[i];

    rws[i] = rws[N - i - 1];
    rws[N - i - 1] = t;

    t = iws[i];

    iws[i] = -iws[N - i - 1];
    iws[N - i - 1] = -t;
  }
};
```

```
FFTM.prototype.normalize13b = function normalize13b (ws, N) {
  var carry = 0;
  for (var i = 0; i < N / 2; i++) {
    var w = Math.round(ws[2 * i + 1] / N) * 0x2000 +
      Math.round(ws[2 * i] / N) +
      carry;

    ws[i] = w & 0x3ffffff;

    if (w < 0x4000000) {
      carry = 0;
    } else {
      carry = w / 0x4000000 | 0;
    }
  }

  return ws;
};

FFTM.prototype.convert13b = function convert13b (ws, len, rws, N) {
  var carry = 0;
  for (var i = 0; i < len; i++) {
    carry = carry + (ws[i] | 0);
```

```
    rws[2 * i] = carry & 0x1fff; carry = carry >>> 13;
    rws[2 * i + 1] = carry & 0x1fff; carry = carry >>> 13;
  }

  // Pad with zeroes
  for (i = 2 * len; i < N; ++i) {
    rws[i] = 0;
  }

  assert(carry === 0);
  assert((carry & ~0x1fff) === 0);
};

FFTM.prototype.stub = function stub (N) {
  var ph = new Array(N);
  for (var i = 0; i < N; i++) {
    ph[i] = 0;
  }

  return ph;
};

FFTM.prototype.mulp = function mulp (x, y, out) {
```

```
var N = 2 * this.guessLen13b(x.length, y.length);

var rbt = this.makeRBT(N);

var _ = this.stub(N);

var rws = new Array(N);
var rwst = new Array(N);
var iwst = new Array(N);

var nrws = new Array(N);
var nrwst = new Array(N);
var niwst = new Array(N);

var rmws = out.words;
rmws.length = N;

this.convert13b(x.words, x.length, rws, N);
this.convert13b(y.words, y.length, nrws, N);

this.transform(rws, _, rwst, iwst, N, rbt);
this.transform(nrws, _, nrwst, niwst, N, rbt);

for (var i = 0; i < N; i++) {
```

```
    var rx = rwst[i] * nrwst[i] - iwst[i] * niwst[i];
    iwst[i] = rwst[i] * niwst[i] + iwst[i] * nrwst[i];
    rwst[i] = rx;
  }

  this.conjugate(rwst, iwst, N);
  this.transform(rwst, iwst, rmws, _, N, rbt);
  this.conjugate(rmws, _, N);
  this.normalize13b(rmws, N);

  out.negative = x.negative ^ y.negative;
  out.length = x.length + y.length;
  return out.strip();
};

// Multiply `this` by `num`
BN.prototype.mul = function mul (num) {
  var out = new BN(null);
  out.words = new Array(this.length + num.length);
  return this.mulTo(num, out);
};

// Multiply employing FFT
BN.prototype.mulf = function mulf (num) {
```

```
  var out = new BN(null);
  out.words = new Array(this.length + num.length);
  return jumboMulTo(this, num, out);
};

// In-place Multiplication
BN.prototype.imul = function imul (num) {
  return this.clone().mulTo(num, this);
};

BN.prototype.imuln = function imuln (num) {
  assert(typeof num === 'number');
  assert(num < 0x4000000);

  // Carry
  var carry = 0;
  for (var i = 0; i < this.length; i++) {
    var w = (this.words[i] | 0) * num;
    var lo = (w & 0x3ffffff) + (carry & 0x3ffffff);
    carry >>= 26;
    carry += (w / 0x4000000) | 0;
    // NOTE: lo is 27bit maximum
    carry += lo >>> 26;
    this.words[i] = lo & 0x3ffffff;
```

```
  }

  if (carry !== 0) {
    this.words[i] = carry;
    this.length++;
  }

  return this;
};

BN.prototype.muln = function muln (num) {
  return this.clone().imuln(num);
};

// `this` * `this`
BN.prototype.sqr = function sqr () {
  return this.mul(this);
};

// `this` * `this` in-place
BN.prototype.isqr = function isqr () {
  return this.imul(this.clone());
};
```

```
// Math.pow(`this`, `num`)
BN.prototype.pow = function pow (num) {
  var w = toBitArray(num);
  if (w.length === 0) return new BN(1);

  // Skip leading zeroes
  var res = this;
  for (var i = 0; i < w.length; i++, res = res.sqr()) {
    if (w[i] !== 0) break;
  }

  if (++i < w.length) {
    for (var q = res.sqr(); i < w.length; i++, q = q.sqr()) {
      if (w[i] === 0) continue;

      res = res.mul(q);
    }
  }

  return res;
};

// Shift-left in-place
BN.prototype.iushln = function iushln (bits) {
```

```
assert(typeof bits === 'number' && bits >= 0);
var r = bits % 26;
var s = (bits - r) / 26;
var carryMask = (0x3ffffff >>> (26 - r)) << (26 - r);
var i;

if (r !== 0) {
  var carry = 0;

  for (i = 0; i < this.length; i++) {
    var newCarry = this.words[i] & carryMask;
    var c = ((this.words[i] | 0) - newCarry) << r;
    this.words[i] = c | carry;
    carry = newCarry >>> (26 - r);
  }

  if (carry) {
    this.words[i] = carry;
    this.length++;
  }
}

if (s !== 0) {
  for (i = this.length - 1; i >= 0; i--) {
```

```
      this.words[i + s] = this.words[i];
    }

    for (i = 0; i < s; i++) {
      this.words[i] = 0;
    }

    this.length += s;
  }

  return this.strip();
};

BN.prototype.ishln = function ishln (bits) {
  // TODO(indutny): implement me
  assert(this.negative === 0);
  return this.iushln(bits);
};

// Shift-right in-place
// NOTE: `hint` is a lowest bit before trailing zeroes
// NOTE: if `extended` is present - it will be filled with destroyed bits
BN.prototype.iushrn = function iushrn (bits, hint, extended) {
  assert(typeof bits === 'number' && bits >= 0);
```

```
var h;
if (hint) {
  h = (hint - (hint % 26)) / 26;
} else {
  h = 0;
}

var r = bits % 26;
var s = Math.min((bits - r) / 26, this.length);
var mask = 0x3ffffff ^ ((0x3ffffff >>> r) << r);
var maskedWords = extended;

h -= s;
h = Math.max(0, h);

// Extended mode, copy masked part
if (maskedWords) {
  for (var i = 0; i < s; i++) {
    maskedWords.words[i] = this.words[i];
  }
  maskedWords.length = s;
}

if (s === 0) {
```

```
  // No-op, we should not move anything at all
} else if (this.length > s) {
  this.length -= s;
  for (i = 0; i < this.length; i++) {
    this.words[i] = this.words[i + s];
  }
} else {
  this.words[0] = 0;
  this.length = 1;
}

var carry = 0;
for (i = this.length - 1; i >= 0 && (carry !== 0 || i >= h); i--) {
  var word = this.words[i] | 0;
  this.words[i] = (carry << (26 - r)) | (word >>> r);
  carry = word & mask;
}

// Push carried bits as a mask
if (maskedWords && carry !== 0) {
  maskedWords.words[maskedWords.length++] = carry;
}

if (this.length === 0) {
```

```
    this.words[0] = 0;
    this.length = 1;
  }

  return this.strip();
};

BN.prototype.ishrn = function ishrn (bits, hint, extended) {
  // TODO(indutny): implement me
  assert(this.negative === 0);
  return this.iushrn(bits, hint, extended);
};

// Shift-left
BN.prototype.shln = function shln (bits) {
  return this.clone().ishln(bits);
};

BN.prototype.ushln = function ushln (bits) {
  return this.clone().iushln(bits);
};

// Shift-right
BN.prototype.shrn = function shrn (bits) {
```

```
  return this.clone().ishrn(bits);
};

BN.prototype.ushrn = function ushrn (bits) {
  return this.clone().iushrn(bits);
};

// Test if n bit is set
BN.prototype.testn = function testn (bit) {
  assert(typeof bit === 'number' && bit >= 0);
  var r = bit % 26;
  var s = (bit - r) / 26;
  var q = 1 << r;

  // Fast case: bit is much higher than all existing words
  if (this.length <= s) return false;

  // Check bit and return
  var w = this.words[s];

  return !!(w & q);
};

// Return only lowers bits of number (in-place)
```

```
BN.prototype.imaskn = function imaskn (bits) {
  assert(typeof bits === 'number' && bits >= 0);
  var r = bits % 26;
  var s = (bits - r) / 26;

  assert(this.negative === 0, 'imaskn works only with positive numbers');

  if (this.length <= s) {
    return this;
  }

  if (r !== 0) {
    s++;
  }
  this.length = Math.min(s, this.length);

  if (r !== 0) {
    var mask = 0x3ffffff ^ ((0x3ffffff >>> r) << r);
    this.words[this.length - 1] &= mask;
  }

  return this.strip();
};
```

```
// Return only lowers bits of number
BN.prototype.maskn = function maskn (bits) {
  return this.clone().imaskn(bits);
};

// Add plain number `num` to `this`
BN.prototype.iaddn = function iaddn (num) {
  assert(typeof num === 'number');
  assert(num < 0x4000000);
  if (num < 0) return this.isubn(-num);

  // Possible sign change
  if (this.negative !== 0) {
    if (this.length === 1 && (this.words[0] | 0) < num) {
      this.words[0] = num - (this.words[0] | 0);
      this.negative = 0;
      return this;
    }

    this.negative = 0;
    this.isubn(num);
    this.negative = 1;
    return this;
  }
```

```
  // Add without checks
  return this._iaddn(num);
};

BN.prototype._iaddn = function _iaddn (num) {
  this.words[0] += num;

  // Carry
  for (var i = 0; i < this.length && this.words[i] >= 0x4000000; i++) {
    this.words[i] -= 0x4000000;
    if (i === this.length - 1) {
      this.words[i + 1] = 1;
    } else {
      this.words[i + 1]++;
    }
  }
  this.length = Math.max(this.length, i + 1);

  return this;
};

// Subtract plain number `num` from `this`
BN.prototype.isubn = function isubn (num) {
```

```
assert(typeof num === 'number');
assert(num < 0x4000000);
if (num < 0) return this.iaddn(-num);

if (this.negative !== 0) {
  this.negative = 0;
  this.iaddn(num);
  this.negative = 1;
  return this;
}

this.words[0] -= num;

if (this.length === 1 && this.words[0] < 0) {
  this.words[0] = -this.words[0];
  this.negative = 1;
} else {
  // Carry
  for (var i = 0; i < this.length && this.words[i] < 0; i++) {
    this.words[i] += 0x4000000;
    this.words[i + 1] -= 1;
  }
}
```

```
  return this.strip();
};

BN.prototype.addn = function addn (num) {
  return this.clone().iaddn(num);
};

BN.prototype.subn = function subn (num) {
  return this.clone().isubn(num);
};

BN.prototype.iabs = function iabs () {
  this.negative = 0;

  return this;
};

BN.prototype.abs = function abs () {
  return this.clone().iabs();
};

BN.prototype._ishlnsubmul = function _ishlnsubmul (num, mul, shift) {
  var len = num.length + shift;
  var i;
```

```
this._expand(len);

var w;
var carry = 0;
for (i = 0; i < num.length; i++) {
  w = (this.words[i + shift] | 0) + carry;
  var right = (num.words[i] | 0) * mul;
  w -= right & 0x3ffffff;
  carry = (w >> 26) - ((right / 0x4000000) | 0);
  this.words[i + shift] = w & 0x3ffffff;
}
for (; i < this.length - shift; i++) {
  w = (this.words[i + shift] | 0) + carry;
  carry = w >> 26;
  this.words[i + shift] = w & 0x3ffffff;
}

if (carry === 0) return this.strip();

// Subtraction overflow
assert(carry === -1);
carry = 0;
for (i = 0; i < this.length; i++) {
```

```
    w = -(this.words[i] | 0) + carry;
    carry = w >> 26;
    this.words[i] = w & 0x3ffffff;
  }
  this.negative = 1;

  return this.strip();
};

BN.prototype._wordDiv = function _wordDiv (num, mode) {
  var shift = this.length - num.length;

  var a = this.clone();
  var b = num;

  // Normalize
  var bhi = b.words[b.length - 1] | 0;
  var bhiBits = this._countBits(bhi);
  shift = 26 - bhiBits;
  if (shift !== 0) {
    b = b.ushln(shift);
    a.iushln(shift);
    bhi = b.words[b.length - 1] | 0;
  }
```

```
// Initialize quotient
var m = a.length - b.length;
var q;

if (mode !== 'mod') {
  q = new BN(null);
  q.length = m + 1;
  q.words = new Array(q.length);
  for (var i = 0; i < q.length; i++) {
    q.words[i] = 0;
  }
}

var diff = a.clone()._ishlnsubmul(b, 1, m);
if (diff.negative === 0) {
  a = diff;
  if (q) {
    q.words[m] = 1;
  }
}

for (var j = m - 1; j >= 0; j--) {
  var qj = (a.words[b.length + j] | 0) * 0x4000000 +
```

```
      (a.words[b.length + j - 1] | 0);

    // NOTE: (qj / bhi) is (0x3ffffff * 0x4000000 + 0x3ffffff) / 0x2000000 max
    // (0x7ffffff)
    qj = Math.min((qj / bhi) | 0, 0x3ffffff);

    a._ishlnsubmul(b, qj, j);
    while (a.negative !== 0) {
      qj--;
      a.negative = 0;
      a._ishlnsubmul(b, 1, j);
      if (!a.isZero()) {
        a.negative ^= 1;
      }
    }
    if (q) {
      q.words[j] = qj;
    }
  }
  if (q) {
    q.strip();
  }
  a.strip();
```

```
  // Denormalize
  if (mode !== 'div' && shift !== 0) {
    a.iushrn(shift);
  }

  return {
    div: q || null,
    mod: a
  };
};

// NOTE: 1) `mode` can be set to `mod` to request mod only,
//       to `div` to request div only, or be absent to
//       request both div & mod
//       2) `positive` is true if unsigned mod is requested
BN.prototype.divmod = function divmod (num, mode, positive) {
  assert(!num.isZero());

  if (this.isZero()) {
    return {
      div: new BN(0),
      mod: new BN(0)
    };
  }
```

```
var div, mod, res;
if (this.negative !== 0 && num.negative === 0) {
  res = this.neg().divmod(num, mode);

  if (mode !== 'mod') {
    div = res.div.neg();
  }

  if (mode !== 'div') {
    mod = res.mod.neg();
    if (positive && mod.negative !== 0) {
      mod.iadd(num);
    }
  }

  return {
    div: div,
    mod: mod
  };
}

if (this.negative === 0 && num.negative !== 0) {
  res = this.divmod(num.neg(), mode);
```

```
    if (mode !== 'mod') {
      div = res.div.neg();
    }

    return {
      div: div,
      mod: res.mod
    };
  }

  if ((this.negative & num.negative) !== 0) {
    res = this.neg().divmod(num.neg(), mode);

    if (mode !== 'div') {
      mod = res.mod.neg();
      if (positive && mod.negative !== 0) {
        mod.isub(num);
      }
    }

    return {
      div: res.div,
      mod: mod
```

```
  };
}

// Both numbers are positive at this point

// Strip both numbers to approximate shift value
if (num.length > this.length || this.cmp(num) < 0) {
  return {
    div: new BN(0),
    mod: this
  };
}

// Very short reduction
if (num.length === 1) {
  if (mode === 'div') {
    return {
      div: this.divn(num.words[0]),
      mod: null
    };
  }

  if (mode === 'mod') {
    return {
```

```
        div: null,
        mod: new BN(this.modn(num.words[0]))
      };
    }

    return {
      div: this.divn(num.words[0]),
      mod: new BN(this.modn(num.words[0]))
    };
  }

  return this._wordDiv(num, mode);
};

// Find `this` / `num`
BN.prototype.div = function div (num) {
  return this.divmod(num, 'div', false).div;
};

// Find `this` % `num`
BN.prototype.mod = function mod (num) {
  return this.divmod(num, 'mod', false).mod;
};
```

```
BN.prototype.umod = function umod (num) {
  return this.divmod(num, 'mod', true).mod;
};

// Find Round(`this` / `num`)
BN.prototype.divRound = function divRound (num) {
  var dm = this.divmod(num);

  // Fast case - exact division
  if (dm.mod.isZero()) return dm.div;

  var mod = dm.div.negative !== 0 ? dm.mod.isub(num) : dm.mod;

  var half = num.ushrn(1);
  var r2 = num.andln(1);
  var cmp = mod.cmp(half);

  // Round down
  if (cmp < 0 || r2 === 1 && cmp === 0) return dm.div;

  // Round up
  return dm.div.negative !== 0 ? dm.div.isubn(1) : dm.div.iaddn(1);
};
```

```
BN.prototype.modn = function modn (num) {
  assert(num <= 0x3ffffff);
  var p = (1 << 26) % num;

  var acc = 0;
  for (var i = this.length - 1; i >= 0; i--) {
    acc = (p * acc + (this.words[i] | 0)) % num;
  }

  return acc;
};

// In-place division by number
BN.prototype.idivn = function idivn (num) {
  assert(num <= 0x3ffffff);

  var carry = 0;
  for (var i = this.length - 1; i >= 0; i--) {
    var w = (this.words[i] | 0) + carry * 0x4000000;
    this.words[i] = (w / num) | 0;
    carry = w % num;
  }

  return this.strip();
```

```
};

BN.prototype.divn = function divn (num) {
  return this.clone().idivn(num);
};

BN.prototype.egcd = function egcd (p) {
  assert(p.negative === 0);
  assert(!p.isZero());

  var x = this;
  var y = p.clone();

  if (x.negative !== 0) {
    x = x.umod(p);
  } else {
    x = x.clone();
  }

  // A * x + B * y = x
  var A = new BN(1);
  var B = new BN(0);

  // C * x + D * y = y
```

```
var C = new BN(0);
var D = new BN(1);

var g = 0;

while (x.isEven() && y.isEven()) {
  x.iushrn(1);
  y.iushrn(1);
  ++g;
}

var yp = y.clone();
var xp = x.clone();

while (!x.isZero()) {
  for (var i = 0, im = 1; (x.words[0] & im) === 0 && i < 26; ++i, im <<= 1);
  if (i > 0) {
    x.iushrn(i);
    while (i-- > 0) {
      if (A.isOdd() || B.isOdd()) {
        A.iadd(yp);
        B.isub(xp);
      }
```

```
    A.iushrn(1);
    B.iushrn(1);
  }
}

for (var j = 0, jm = 1; (y.words[0] & jm) === 0 && j < 26; ++j, jm <<= 1);
if (j > 0) {
  y.iushrn(j);
  while (j-- > 0) {
    if (C.isOdd() || D.isOdd()) {
      C.iadd(yp);
      D.isub(xp);
    }

    C.iushrn(1);
    D.iushrn(1);
  }
}

if (x.cmp(y) >= 0) {
  x.isub(y);
  A.isub(C);
  B.isub(D);
} else {
```

```
      y.isub(x);
      C.isub(A);
      D.isub(B);
    }
  }

  return {
    a: C,
    b: D,
    gcd: y.iushln(g)
  };
};

// This is reduced incarnation of the binary EEA
// above, designated to invert members of the
// _prime_ fields F(p) at a maximal speed
BN.prototype._invmp = function _invmp (p) {
  assert(p.negative === 0);
  assert(!p.isZero());

  var a = this;
  var b = p.clone();

  if (a.negative !== 0) {
```

```
  a = a.umod(p);
} else {
  a = a.clone();
}

var x1 = new BN(1);
var x2 = new BN(0);

var delta = b.clone();

while (a.cmpn(1) > 0 && b.cmpn(1) > 0) {
  for (var i = 0, im = 1; (a.words[0] & im) === 0 && i < 26; ++i, im <<= 1);
  if (i > 0) {
    a.iushrn(i);
    while (i-- > 0) {
      if (x1.isOdd()) {
        x1.iadd(delta);
      }

      x1.iushrn(1);
    }
  }

  for (var j = 0, jm = 1; (b.words[0] & jm) === 0 && j < 26; ++j, jm <<= 1);
```

```
  if (j > 0) {
    b.iushrn(j);
    while (j-- > 0) {
      if (x2.isOdd()) {
        x2.iadd(delta);
      }

      x2.iushrn(1);
    }
  }

  if (a.cmp(b) >= 0) {
    a.isub(b);
    x1.isub(x2);
  } else {
    b.isub(a);
    x2.isub(x1);
  }
}

var res;
if (a.cmpn(1) === 0) {
  res = x1;
} else {
```

```
    res = x2;
  }

  if (res.cmpn(0) < 0) {
    res.iadd(p);
  }

  return res;
};

BN.prototype.gcd = function gcd (num) {
  if (this.isZero()) return num.abs();
  if (num.isZero()) return this.abs();

  var a = this.clone();
  var b = num.clone();
  a.negative = 0;
  b.negative = 0;

  // Remove common factor of two
  for (var shift = 0; a.isEven() && b.isEven(); shift++) {
    a.iushrn(1);
    b.iushrn(1);
  }
```

```
  do {
    while (a.isEven()) {
      a.iushrn(1);
    }
    while (b.isEven()) {
      b.iushrn(1);
    }

    var r = a.cmp(b);
    if (r < 0) {
      // Swap `a` and `b` to make `a` always bigger than `b`
      var t = a;
      a = b;
      b = t;
    } else if (r === 0 || b.cmpn(1) === 0) {
      break;
    }

    a.isub(b);
  } while (true);

  return b.iushln(shift);
};
```

```
// Invert number in the field F(num)
BN.prototype.invm = function invm (num) {
  return this.egcd(num).a.umod(num);
};

BN.prototype.isEven = function isEven () {
  return (this.words[0] & 1) === 0;
};

BN.prototype.isOdd = function isOdd () {
  return (this.words[0] & 1) === 1;
};

// And first word and num
BN.prototype.andln = function andln (num) {
  return this.words[0] & num;
};

// Increment at the bit position in-line
BN.prototype.bincn = function bincn (bit) {
  assert(typeof bit === 'number');
  var r = bit % 26;
  var s = (bit - r) / 26;
```

```
  var q = 1 << r;

  // Fast case: bit is much higher than all existing words
  if (this.length <= s) {
    this._expand(s + 1);
    this.words[s] |= q;
    return this;
  }

  // Add bit and propagate, if needed
  var carry = q;
  for (var i = s; carry !== 0 && i < this.length; i++) {
    var w = this.words[i] | 0;
    w += carry;
    carry = w >>> 26;
    w &= 0x3ffffff;
    this.words[i] = w;
  }
  if (carry !== 0) {
    this.words[i] = carry;
    this.length++;
  }
  return this;
};
```

```
BN.prototype.isZero = function isZero () {
  return this.length === 1 && this.words[0] === 0;
};

BN.prototype.cmpn = function cmpn (num) {
  var negative = num < 0;

  if (this.negative !== 0 && !negative) return -1;
  if (this.negative === 0 && negative) return 1;

  this.strip();

  var res;
  if (this.length > 1) {
    res = 1;
  } else {
    if (negative) {
      num = -num;
    }

    assert(num <= 0x3ffffff, 'Number is too big');

    var w = this.words[0] | 0;
```

```
    res = w === num ? 0 : w < num ? -1 : 1;
  }
  if (this.negative !== 0) return -res | 0;
  return res;
};

// Compare two numbers and return:
// 1 - if `this` > `num`
// 0 - if `this` == `num`
// -1 - if `this` < `num`
BN.prototype.cmp = function cmp (num) {
  if (this.negative !== 0 && num.negative === 0) return -1;
  if (this.negative === 0 && num.negative !== 0) return 1;

  var res = this.ucmp(num);
  if (this.negative !== 0) return -res | 0;
  return res;
};

// Unsigned comparison
BN.prototype.ucmp = function ucmp (num) {
  // At this point both numbers have the same sign
  if (this.length > num.length) return 1;
  if (this.length < num.length) return -1;
```

```
  var res = 0;
  for (var i = this.length - 1; i >= 0; i--) {
    var a = this.words[i] | 0;
    var b = num.words[i] | 0;

    if (a === b) continue;
    if (a < b) {
      res = -1;
    } else if (a > b) {
      res = 1;
    }
    break;
  }
  return res;
};

BN.prototype.gtn = function gtn (num) {
  return this.cmpn(num) === 1;
};

BN.prototype.gt = function gt (num) {
  return this.cmp(num) === 1;
};
```

```
BN.prototype.gten = function gten (num) {
  return this.cmpn(num) >= 0;
};

BN.prototype.gte = function gte (num) {
  return this.cmp(num) >= 0;
};

BN.prototype.ltn = function ltn (num) {
  return this.cmpn(num) === -1;
};

BN.prototype.lt = function lt (num) {
  return this.cmp(num) === -1;
};

BN.prototype.lten = function lten (num) {
  return this.cmpn(num) <= 0;
};

BN.prototype.lte = function lte (num) {
  return this.cmp(num) <= 0;
};
```

```
BN.prototype.eqn = function eqn (num) {
  return this.cmpn(num) === 0;
};

BN.prototype.eq = function eq (num) {
  return this.cmp(num) === 0;
};

//
// A reduce context, could be using montgomery or something better,
depending
// on the `m` itself.
//
BN.red = function red (num) {
  return new Red(num);
};

BN.prototype.toRed = function toRed (ctx) {
  assert(!this.red, 'Already a number in reduction context');
  assert(this.negative === 0, 'red works only with positives');
  return ctx.convertTo(this)._forceRed(ctx);
};

BN.prototype.fromRed = function fromRed () {
```

```
  assert(this.red, 'fromRed works only with numbers in reduction context');
  return this.red.convertFrom(this);
};

BN.prototype._forceRed = function _forceRed (ctx) {
  this.red = ctx;
  return this;
};

BN.prototype.forceRed = function forceRed (ctx) {
  assert(!this.red, 'Already a number in reduction context');
  return this._forceRed(ctx);
};

BN.prototype.redAdd = function redAdd (num) {
  assert(this.red, 'redAdd works only with red numbers');
  return this.red.add(this, num);
};

BN.prototype.redIAdd = function redIAdd (num) {
  assert(this.red, 'redIAdd works only with red numbers');
  return this.red.iadd(this, num);
};
```

```
BN.prototype.redSub = function redSub (num) {
  assert(this.red, 'redSub works only with red numbers');
  return this.red.sub(this, num);
};

BN.prototype.redISub = function redISub (num) {
  assert(this.red, 'redISub works only with red numbers');
  return this.red.isub(this, num);
};

BN.prototype.redShl = function redShl (num) {
  assert(this.red, 'redShl works only with red numbers');
  return this.red.shl(this, num);
};

BN.prototype.redMul = function redMul (num) {
  assert(this.red, 'redMul works only with red numbers');
  this.red._verify2(this, num);
  return this.red.mul(this, num);
};

BN.prototype.redIMul = function redIMul (num) {
  assert(this.red, 'redMul works only with red numbers');
  this.red._verify2(this, num);
```

```
  return this.red.imul(this, num);
};

BN.prototype.redSqr = function redSqr () {
  assert(this.red, 'redSqr works only with red numbers');
  this.red._verify1(this);
  return this.red.sqr(this);
};

BN.prototype.redISqr = function redISqr () {
  assert(this.red, 'redISqr works only with red numbers');
  this.red._verify1(this);
  return this.red.isqr(this);
};

// Square root over p
BN.prototype.redSqrt = function redSqrt () {
  assert(this.red, 'redSqrt works only with red numbers');
  this.red._verify1(this);
  return this.red.sqrt(this);
};

BN.prototype.redInvm = function redInvm () {
  assert(this.red, 'redInvm works only with red numbers');
```

```
  this.red._verify1(this);
  return this.red.invm(this);
};

// Return negative clone of `this` % `red modulo`
BN.prototype.redNeg = function redNeg () {
  assert(this.red, 'redNeg works only with red numbers');
  this.red._verify1(this);
  return this.red.neg(this);
};

BN.prototype.redPow = function redPow (num) {
  assert(this.red && !num.red, 'redPow(normalNum)');
  this.red._verify1(this);
  return this.red.pow(this, num);
};

// Prime numbers with efficient reduction
var primes = {
  k256: null,
  p224: null,
  p192: null,
  p25519: null
};
```

```
// Pseudo-Mersenne prime
function MPrime (name, p) {
  // P = 2 ^ N - K
  this.name = name;
  this.p = new BN(p, 16);
  this.n = this.p.bitLength();
  this.k = new BN(1).iushln(this.n).isub(this.p);

  this.tmp = this._tmp();
}

MPrime.prototype._tmp = function _tmp () {
  var tmp = new BN(null);
  tmp.words = new Array(Math.ceil(this.n / 13));
  return tmp;
};

MPrime.prototype.ireduce = function ireduce (num) {
  // Assumes that `num` is less than `P^2`
  // num = HI * (2 ^ N - K) + HI * K + LO = HI * K + LO (mod P)
  var r = num;
  var rlen;
```

```
  do {
    this.split(r, this.tmp);
    r = this.imulK(r);
    r = r.iadd(this.tmp);
    rlen = r.bitLength();
  } while (rlen > this.n);

  var cmp = rlen < this.n ? -1 : r.ucmp(this.p);
  if (cmp === 0) {
    r.words[0] = 0;
    r.length = 1;
  } else if (cmp > 0) {
    r.isub(this.p);
  } else {
    r.strip();
  }

  return r;
};

MPrime.prototype.split = function split (input, out) {
  input.iushrn(this.n, 0, out);
};
```

```
MPrime.prototype.imulK = function imulK (num) {
  return num.imul(this.k);
};

function K256 () {
  MPrime.call(
    this,
    'k256',
    'ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffe fffffc2f');
}
inherits(K256, MPrime);

K256.prototype.split = function split (input, output) {
  // 256 = 9 * 26 + 22
  var mask = 0x3fffff;

  var outLen = Math.min(input.length, 9);
  for (var i = 0; i < outLen; i++) {
    output.words[i] = input.words[i];
  }
  output.length = outLen;

  if (input.length <= 9) {
    input.words[0] = 0;
```

```
    input.length = 1;
    return;
  }

  // Shift by 9 limbs
  var prev = input.words[9];
  output.words[output.length++] = prev & mask;

  for (i = 10; i < input.length; i++) {
    var next = input.words[i] | 0;
    input.words[i - 10] = ((next & mask) << 4) | (prev >>> 22);
    prev = next;
  }
  prev >>>= 22;
  input.words[i - 10] = prev;
  if (prev === 0 && input.length > 10) {
    input.length -= 10;
  } else {
    input.length -= 9;
  }
};

K256.prototype.imulK = function imulK (num) {
  // K = 0x1000003d1 = [ 0x40, 0x3d1 ]
```

```
  num.words[num.length] = 0;
  num.words[num.length + 1] = 0;
  num.length += 2;

  // bounded at: 0x40 * 0x3ffffff + 0x3d0 = 0x100000390
  var lo = 0;
  for (var i = 0; i < num.length; i++) {
    var w = num.words[i] | 0;
    lo += w * 0x3d1;
    num.words[i] = lo & 0x3ffffff;
    lo = w * 0x40 + ((lo / 0x4000000) | 0);
  }

  // Fast length reduction
  if (num.words[num.length - 1] === 0) {
    num.length--;
    if (num.words[num.length - 1] === 0) {
      num.length--;
    }
  }
  return num;
};

function P224 () {
```

```
  MPrime.call(
    this,
    'p224',
    'ffffffff ffffffff ffffffff ffffffff 00000000 00000000 00000001');
}
inherits(P224, MPrime);

function P192 () {
  MPrime.call(
    this,
    'p192',
    'ffffffff ffffffff ffffffff fffffffe ffffffff ffffffff');
}
inherits(P192, MPrime);

function P25519 () {
  // 2 ^ 255 - 19
  MPrime.call(
    this,
    '25519',
    '7fffffffffffffff ffffffffffffffff ffffffffffffffff ffffffffffffffed');
}
inherits(P25519, MPrime);
```

```
P25519.prototype.imulK = function imulK (num) {
  // K = 0x13
  var carry = 0;
  for (var i = 0; i < num.length; i++) {
    var hi = (num.words[i] | 0) * 0x13 + carry;
    var lo = hi & 0x3ffffff;
    hi >>>= 26;

    num.words[i] = lo;
    carry = hi;
  }
  if (carry !== 0) {
    num.words[num.length++] = carry;
  }
  return num;
};

// Exported mostly for testing purposes, use plain name instead
BN._prime = function prime (name) {
  // Cached version of prime
  if (primes[name]) return primes[name];

  var prime;
  if (name === 'k256') {
```

```
    prime = new K256();
  } else if (name === 'p224') {
    prime = new P224();
  } else if (name === 'p192') {
    prime = new P192();
  } else if (name === 'p25519') {
    prime = new P25519();
  } else {
    throw new Error('Unknown prime ' + name);
  }
  primes[name] = prime;

  return prime;
};

//
// Base reduction engine
//
function Red (m) {
  if (typeof m === 'string') {
    var prime = BN._prime(m);
    this.m = prime.p;
    this.prime = prime;
  } else {
```

```
    assert(m.gtn(1), 'modulus must be greater than 1');
    this.m = m;
    this.prime = null;
  }
}

Red.prototype._verify1 = function _verify1 (a) {
  assert(a.negative === 0, 'red works only with positives');
  assert(a.red, 'red works only with red numbers');
};

Red.prototype._verify2 = function _verify2 (a, b) {
  assert((a.negative | b.negative) === 0, 'red works only with positives');
  assert(a.red && a.red === b.red,
    'red works only with red numbers');
};

Red.prototype.imod = function imod (a) {
  if (this.prime) return this.prime.ireduce(a)._forceRed(this);
  return a.umod(this.m)._forceRed(this);
};

Red.prototype.neg = function neg (a) {
  if (a.isZero()) {
```

```
    return a.clone();
  }

  return this.m.sub(a)._forceRed(this);
};

Red.prototype.add = function add (a, b) {
  this._verify2(a, b);

  var res = a.add(b);
  if (res.cmp(this.m) >= 0) {
    res.isub(this.m);
  }
  return res._forceRed(this);
};

Red.prototype.iadd = function iadd (a, b) {
  this._verify2(a, b);

  var res = a.iadd(b);
  if (res.cmp(this.m) >= 0) {
    res.isub(this.m);
  }
  return res;
```

```
};

Red.prototype.sub = function sub (a, b) {
  this._verify2(a, b);

  var res = a.sub(b);
  if (res.cmpn(0) < 0) {
    res.iadd(this.m);
  }
  return res._forceRed(this);
};

Red.prototype.isub = function isub (a, b) {
  this._verify2(a, b);

  var res = a.isub(b);
  if (res.cmpn(0) < 0) {
    res.iadd(this.m);
  }
  return res;
};

Red.prototype.shl = function shl (a, num) {
  this._verify1(a);
```

```
  return this.imod(a.ushln(num));
};

Red.prototype.imul = function imul (a, b) {
  this._verify2(a, b);
  return this.imod(a.imul(b));
};

Red.prototype.mul = function mul (a, b) {
  this._verify2(a, b);
  return this.imod(a.mul(b));
};

Red.prototype.isqr = function isqr (a) {
  return this.imul(a, a.clone());
};

Red.prototype.sqr = function sqr (a) {
  return this.mul(a, a);
};

Red.prototype.sqrt = function sqrt (a) {
  if (a.isZero()) return a.clone();
```

```
var mod3 = this.m.andln(3);
assert(mod3 % 2 === 1);

// Fast case
if (mod3 === 3) {
  var pow = this.m.add(new BN(1)).iushrn(2);
  return this.pow(a, pow);
}

// Tonelli-Shanks algorithm (Totally unoptimized and slow)
//
// Find Q and S, that Q * 2 ^ S = (P - 1)
var q = this.m.subn(1);
var s = 0;
while (!q.isZero() && q.andln(1) === 0) {
  s++;
  q.iushrn(1);
}
assert(!q.isZero());

var one = new BN(1).toRed(this);
var nOne = one.redNeg();

// Find quadratic non-residue
```

```
// NOTE: Max is such because of generalized Riemann hypothesis.
var lpow = this.m.subn(1).iushrn(1);
var z = this.m.bitLength();
z = new BN(2 * z * z).toRed(this);

while (this.pow(z, lpow).cmp(nOne) !== 0) {
  z.redIAdd(nOne);
}

var c = this.pow(z, q);
var r = this.pow(a, q.addn(1).iushrn(1));
var t = this.pow(a, q);
var m = s;
while (t.cmp(one) !== 0) {
  var tmp = t;
  for (var i = 0; tmp.cmp(one) !== 0; i++) {
    tmp = tmp.redSqr();
  }
  assert(i < m);
  var b = this.pow(c, new BN(1).iushln(m - i - 1));

  r = r.redMul(b);
  c = b.redSqr();
  t = t.redMul(c);
```

```
    m = i;
  }

  return r;
};

Red.prototype.invm = function invm (a) {
  var inv = a._invmp(this.m);
  if (inv.negative !== 0) {
    inv.negative = 0;
    return this.imod(inv).redNeg();
  } else {
    return this.imod(inv);
  }
};

Red.prototype.pow = function pow (a, num) {
  if (num.isZero()) return new BN(1);
  if (num.cmpn(1) === 0) return a.clone();

  var windowSize = 4;
  var wnd = new Array(1 << windowSize);
  wnd[0] = new BN(1).toRed(this);
  wnd[1] = a;
```

```
for (var i = 2; i < wnd.length; i++) {
  wnd[i] = this.mul(wnd[i - 1], a);
}

var res = wnd[0];
var current = 0;
var currentLen = 0;
var start = num.bitLength() % 26;
if (start === 0) {
  start = 26;
}

for (i = num.length - 1; i >= 0; i--) {
  var word = num.words[i];
  for (var j = start - 1; j >= 0; j--) {
    var bit = (word >> j) & 1;
    if (res !== wnd[0]) {
      res = this.sqr(res);
    }

    if (bit === 0 && current === 0) {
      currentLen = 0;
      continue;
    }
```

```
      current <<= 1;
      current |= bit;
      currentLen++;
      if (currentLen !== windowSize && (i !== 0 || j !== 0)) continue;

      res = this.mul(res, wnd[current]);
      currentLen = 0;
      current = 0;
    }
    start = 26;
  }

  return res;
};

Red.prototype.convertTo = function convertTo (num) {
  var r = num.umod(this.m);

  return r === num ? r.clone() : r;
};

Red.prototype.convertFrom = function convertFrom (num) {
  var res = num.clone();
```

```
  res.red = null;
  return res;
};

//
// Montgomery method engine
//

BN.mont = function mont (num) {
  return new Mont(num);
};

function Mont (m) {
  Red.call(this, m);

  this.shift = this.m.bitLength();
  if (this.shift % 26 !== 0) {
    this.shift += 26 - (this.shift % 26);
  }

  this.r = new BN(1).iushln(this.shift);
  this.r2 = this.imod(this.r.sqr());
  this.rinv = this.r._invmp(this.m);
```

```
  this.minv = this.rinv.mul(this.r).isubn(1).div(this.m);
  this.minv = this.minv.umod(this.r);
  this.minv = this.r.sub(this.minv);
}
inherits(Mont, Red);

Mont.prototype.convertTo = function convertTo (num) {
  return this.imod(num.ushln(this.shift));
};

Mont.prototype.convertFrom = function convertFrom (num) {
  var r = this.imod(num.mul(this.rinv));
  r.red = null;
  return r;
};

Mont.prototype.imul = function imul (a, b) {
  if (a.isZero() || b.isZero()) {
    a.words[0] = 0;
    a.length = 1;
    return a;
  }

  var t = a.imul(b);
```

```
  var c = t.maskn(this.shift).mul(this.minv).imaskn(this.shift).mul(this.m);
  var u = t.isub(c).iushrn(this.shift);
  var res = u;

  if (u.cmp(this.m) >= 0) {
    res = u.isub(this.m);
  } else if (u.cmpn(0) < 0) {
    res = u.iadd(this.m);
  }

  return res._forceRed(this);
};

Mont.prototype.mul = function mul (a, b) {
  if (a.isZero() || b.isZero()) return new BN(0)._forceRed(this);

  var t = a.mul(b);
  var c = t.maskn(this.shift).mul(this.minv).imaskn(this.shift).mul(this.m);
  var u = t.isub(c).iushrn(this.shift);
  var res = u;
  if (u.cmp(this.m) >= 0) {
    res = u.isub(this.m);
  } else if (u.cmpn(0) < 0) {
    res = u.iadd(this.m);
```

```
    }

    return res._forceRed(this);
  };

  Mont.prototype.invm = function invm (a) {
    // (AR)^-1 * R^2 = (A^-1 * R^-1) * R^2 = A^-1 * R
    var res = this.imod(a._invmp(this.m).mul(this.r2));
    return res._forceRed(this);
  };
})(typeof module === 'undefined' || module, this);

},{}],5:[function(require,module,exports){

},{}],6:[function(require,module,exports){
/*!
 * The buffer module from node.js, for the browser.
 *
 * @author   Feross Aboukhadijeh <feross@feross.org> <http://feross.org>
 * @license  MIT
 */
/* eslint-disable no-proto */

'use strict'
```

```
var base64 = require('base64-js')
var ieee754 = require('ieee754')

exports.Buffer = Buffer
exports.SlowBuffer = SlowBuffer
exports.INSPECT_MAX_BYTES = 50

var K_MAX_LENGTH = 0x7fffffff
exports.kMaxLength = K_MAX_LENGTH

/**
 * If `Buffer.TYPED_ARRAY_SUPPORT`:
 *   === true    Use Uint8Array implementation (fastest)
 *   === false   Print warning and recommend using `buffer` v4.x which has an
Object
 *               implementation (most compatible, even IE6)
 *
 * Browsers that support typed arrays are IE 10+, Firefox 4+, Chrome 7+, Safari
5.1+,
 * Opera 11.6+, iOS 4.2+.
 *
 * We report that the browser does not support typed arrays if the are not
subclassable
 * using __proto__. Firefox 4-29 lacks support for adding new properties to
`Uint8Array`
```

```
 * (See: https://bugzilla.mozilla.org/show_bug.cgi?id=695438). IE 10 lacks
support
 * for __proto__ and has a buggy typed array implementation.
 */
Buffer.TYPED_ARRAY_SUPPORT = typedArraySupport()

if (!Buffer.TYPED_ARRAY_SUPPORT) {
  console.error(
    'This browser lacks typed array (Uint8Array) support which is required by ' +
    '`buffer` v5.x. Use `buffer` v4.x if you require old browser support.')
}

function typedArraySupport () {
  // Can typed array instances can be augmented?
  try {
    var arr = new Uint8Array(1)
    arr.__proto__ = {__proto__: Uint8Array.prototype, foo: function () { return 42
}}
    return arr.foo() === 42
  } catch (e) {
    return false
  }
}

function createBuffer (length) {
```

```
  if (length > K_MAX_LENGTH) {
    throw new RangeError('Invalid typed array length')
  }
  // Return an augmented `Uint8Array` instance
  var buf = new Uint8Array(length)
  buf.__proto__ = Buffer.prototype
  return buf
}

/**
 * The Buffer constructor returns instances of `Uint8Array` that have their
 * prototype changed to `Buffer.prototype`. Furthermore, `Buffer` is a subclass
of
 * `Uint8Array`, so the returned instances will have all the node `Buffer`
methods
 * and the `Uint8Array` methods. Square bracket notation works as expected --
it
 * returns a single octet.
 *
 * The `Uint8Array` prototype remains unmodified.
 */

function Buffer (arg, encodingOrOffset, length) {
  // Common case.
  if (typeof arg === 'number') {
```

```
    if (typeof encodingOrOffset === 'string') {
      throw new Error(
        'If encoding is specified then the first argument must be a string'
      )
    }
    return allocUnsafe(arg)
  }
  return from(arg, encodingOrOffset, length)
}

// Fix subarray() in ES2016. See: https://github.com/feross/buffer/pull/97
if (typeof Symbol !== 'undefined' && Symbol.species &&
    Buffer[Symbol.species] === Buffer) {
  Object.defineProperty(Buffer, Symbol.species, {
    value: null,
    configurable: true,
    enumerable: false,
    writable: false
  })
}

Buffer.poolSize = 8192 // not used by this implementation

function from (value, encodingOrOffset, length) {
```

```
  if (typeof value === 'number') {
    throw new TypeError('"value" argument must not be a number')
  }

  if (typeof ArrayBuffer !== 'undefined' && value instanceof ArrayBuffer) {
    return fromArrayBuffer(value, encodingOrOffset, length)
  }

  if (typeof value === 'string') {
    return fromString(value, encodingOrOffset)
  }

  return fromObject(value)
}

/**
 * Functionally equivalent to Buffer(arg, encoding) but throws a TypeError
 * if value is a number.
 * Buffer.from(str[, encoding])
 * Buffer.from(array)
 * Buffer.from(buffer)
 * Buffer.from(arrayBuffer[, byteOffset[, length]])
 **/
Buffer.from = function (value, encodingOrOffset, length) {
```

```
  return from(value, encodingOrOffset, length)
}

// Note: Change prototype *after* Buffer.from is defined to workaround
Chrome bug:
// https://github.com/feross/buffer/pull/148
Buffer.prototype.__proto__ = Uint8Array.prototype
Buffer.__proto__ = Uint8Array

function assertSize (size) {
  if (typeof size !== 'number') {
    throw new TypeError('"size" argument must be a number')
  } else if (size < 0) {
    throw new RangeError('"size" argument must not be negative')
  }
}

function alloc (size, fill, encoding) {
  assertSize(size)
  if (size <= 0) {
    return createBuffer(size)
  }
  if (fill !== undefined) {
    // Only pay attention to encoding if it's a string. This
    // prevents accidentally sending in a number that would
```

```
      // be interpretted as a start offset.
      return typeof encoding === 'string'
        ? createBuffer(size).fill(fill, encoding)
        : createBuffer(size).fill(fill)
    }
    return createBuffer(size)
  }

  /**
   * Creates a new filled Buffer instance.
   * alloc(size[, fill[, encoding]])
   **/
  Buffer.alloc = function (size, fill, encoding) {
    return alloc(size, fill, encoding)
  }

  function allocUnsafe (size) {
    assertSize(size)
    return createBuffer(size < 0 ? 0 : checked(size) | 0)
  }

  /**
   * Equivalent to Buffer(num), by default creates a non-zero-filled Buffer
instance.
   * */
```

```
Buffer.allocUnsafe = function (size) {
  return allocUnsafe(size)
}
/**
 * Equivalent to SlowBuffer(num), by default creates a non-zero-filled Buffer
instance.
 */
Buffer.allocUnsafeSlow = function (size) {
  return allocUnsafe(size)
}

function fromString (string, encoding) {
  if (typeof encoding !== 'string' || encoding === '') {
    encoding = 'utf8'
  }

  if (!Buffer.isEncoding(encoding)) {
    throw new TypeError('"encoding" must be a valid string encoding')
  }

  var length = byteLength(string, encoding) | 0
  var buf = createBuffer(length)

  var actual = buf.write(string, encoding)
```

```
  if (actual !== length) {
    // Writing a hex string, for example, that contains invalid characters will
    // cause everything after the first invalid character to be ignored. (e.g.
    // 'abxxcd' will be treated as 'ab')
    buf = buf.slice(0, actual)
  }

  return buf
}

function fromArrayLike (array) {
  var length = array.length < 0 ? 0 : checked(array.length) | 0
  var buf = createBuffer(length)
  for (var i = 0; i < length; i += 1) {
    buf[i] = array[i] & 255
  }
  return buf
}

function fromArrayBuffer (array, byteOffset, length) {
  array.byteLength // this throws if `array` is not a valid ArrayBuffer

  if (byteOffset < 0 || array.byteLength < byteOffset) {
    throw new RangeError('\'offset\' is out of bounds')
```

```
  }

  if (array.byteLength < byteOffset + (length || 0)) {
    throw new RangeError('\'length\' is out of bounds')
  }

  var buf
  if (byteOffset === undefined && length === undefined) {
    buf = new Uint8Array(array)
  } else if (length === undefined) {
    buf = new Uint8Array(array, byteOffset)
  } else {
    buf = new Uint8Array(array, byteOffset, length)
  }

  // Return an augmented `Uint8Array` instance
  buf.__proto__ = Buffer.prototype
  return buf
}

function fromObject (obj) {
  if (Buffer.isBuffer(obj)) {
    var len = checked(obj.length) | 0
    var buf = createBuffer(len)
```

```
    if (buf.length === 0) {
      return buf
    }

    obj.copy(buf, 0, 0, len)
    return buf
  }

  if (obj) {
    if ((typeof ArrayBuffer !== 'undefined' &&
        obj.buffer instanceof ArrayBuffer) || 'length' in obj) {
      if (typeof obj.length !== 'number' || isnan(obj.length)) {
        return createBuffer(0)
      }
      return fromArrayLike(obj)
    }

    if (obj.type === 'Buffer' && Array.isArray(obj.data)) {
      return fromArrayLike(obj.data)
    }
  }

  throw new TypeError('First argument must be a string, Buffer, ArrayBuffer,
Array, or array-like object.')
```

```
}

function checked (length) {
  // Note: cannot use `length < K_MAX_LENGTH` here because that fails when
  // length is NaN (which is otherwise coerced to zero.)
  if (length >= K_MAX_LENGTH) {
    throw new RangeError('Attempt to allocate Buffer larger than maximum ' +
                         'size: 0x' + K_MAX_LENGTH.toString(16) + ' bytes')
  }
  return length | 0
}

function SlowBuffer (length) {
  if (+length != length) { // eslint-disable-line eqeqeq
    length = 0
  }
  return Buffer.alloc(+length)
}

Buffer.isBuffer = function isBuffer (b) {
  return !!(b != null && b._isBuffer)
}

Buffer.compare = function compare (a, b) {
```

```
  if (!Buffer.isBuffer(a) || !Buffer.isBuffer(b)) {
    throw new TypeError('Arguments must be Buffers')
  }

  if (a === b) return 0

  var x = a.length
  var y = b.length

  for (var i = 0, len = Math.min(x, y); i < len; ++i) {
    if (a[i] !== b[i]) {
      x = a[i]
      y = b[i]
      break
    }
  }

  if (x < y) return -1
  if (y < x) return 1
  return 0
}

Buffer.isEncoding = function isEncoding (encoding) {
  switch (String(encoding).toLowerCase()) {
```

```
    case 'hex':
    case 'utf8':
    case 'utf-8':
    case 'ascii':
    case 'latin1':
    case 'binary':
    case 'base64':
    case 'ucs2':
    case 'ucs-2':
    case 'utf16le':
    case 'utf-16le':
      return true
    default:
      return false
  }
}

Buffer.concat = function concat (list, length) {
  if (!Array.isArray(list)) {
    throw new TypeError('"list" argument must be an Array of Buffers')
  }

  if (list.length === 0) {
    return Buffer.alloc(0)
```

```
  }

  var i
  if (length === undefined) {
    length = 0
    for (i = 0; i < list.length; ++i) {
      length += list[i].length
    }
  }

  var buffer = Buffer.allocUnsafe(length)
  var pos = 0
  for (i = 0; i < list.length; ++i) {
    var buf = list[i]
    if (!Buffer.isBuffer(buf)) {
      throw new TypeError('"list" argument must be an Array of Buffers')
    }
    buf.copy(buffer, pos)
    pos += buf.length
  }
  return buffer
}

function byteLength (string, encoding) {
```

```
if (Buffer.isBuffer(string)) {
  return string.length
}
if (typeof ArrayBuffer !== 'undefined' && typeof ArrayBuffer.isView ===
'function' &&
    (ArrayBuffer.isView(string) || string instanceof ArrayBuffer)) {
  return string.byteLength
}
if (typeof string !== 'string') {
  string = '' + string
}

var len = string.length
if (len === 0) return 0

// Use a for loop to avoid recursion
var loweredCase = false
for (;;) {
  switch (encoding) {
    case 'ascii':
    case 'latin1':
    case 'binary':
      return len
    case 'utf8':
    case 'utf-8':
```

```
      case undefined:
        return utf8ToBytes(string).length
      case 'ucs2':
      case 'ucs-2':
      case 'utf16le':
      case 'utf-16le':
        return len * 2
      case 'hex':
        return len >>> 1
      case 'base64':
        return base64ToBytes(string).length
      default:
        if (loweredCase) return utf8ToBytes(string).length // assume utf8
        encoding = ('' + encoding).toLowerCase()
        loweredCase = true
    }
  }
}
Buffer.byteLength = byteLength

function slowToString (encoding, start, end) {
  var loweredCase = false

  // No need to verify that "this.length <= MAX_UINT32" since it's a read-only
```

```
// property of a typed array.

// This behaves neither like String nor Uint8Array in that we set start/end
// to their upper/lower bounds if the value passed is out of range.
// undefined is handled specially as per ECMA-262 6th Edition,
// Section 13.3.3.7 Runtime Semantics: KeyedBindingInitialization.
if (start === undefined || start < 0) {
  start = 0
}
// Return early if start > this.length. Done here to prevent potential uint32
// coercion fail below.
if (start > this.length) {
  return ''
}

if (end === undefined || end > this.length) {
  end = this.length
}

if (end <= 0) {
  return ''
}

// Force coersion to uint32. This will also coerce falsey/NaN values to 0.
```

```
end >>>= 0
start >>>= 0

if (end <= start) {
  return ''
}

if (!encoding) encoding = 'utf8'

while (true) {
  switch (encoding) {
    case 'hex':
      return hexSlice(this, start, end)

    case 'utf8':
    case 'utf-8':
      return utf8Slice(this, start, end)

    case 'ascii':
      return asciiSlice(this, start, end)

    case 'latin1':
    case 'binary':
      return latin1Slice(this, start, end)
```

```
      case 'base64':
        return base64Slice(this, start, end)

      case 'ucs2':
      case 'ucs-2':
      case 'utf16le':
      case 'utf-16le':
        return utf16leSlice(this, start, end)

      default:
        if (loweredCase) throw new TypeError('Unknown encoding: ' + encoding)
        encoding = (encoding + '').toLowerCase()
        loweredCase = true
    }
  }
}

// The property is used by `Buffer.isBuffer` and `is-buffer` (in Safari 5-7) to detect
// Buffer instances.
Buffer.prototype._isBuffer = true

function swap (b, n, m) {
  var i = b[n]
```

```
  b[n] = b[m]
  b[m] = i
}

Buffer.prototype.swap16 = function swap16 () {
  var len = this.length
  if (len % 2 !== 0) {
    throw new RangeError('Buffer size must be a multiple of 16-bits')
  }
  for (var i = 0; i < len; i += 2) {
    swap(this, i, i + 1)
  }
  return this
}

Buffer.prototype.swap32 = function swap32 () {
  var len = this.length
  if (len % 4 !== 0) {
    throw new RangeError('Buffer size must be a multiple of 32-bits')
  }
  for (var i = 0; i < len; i += 4) {
    swap(this, i, i + 3)
    swap(this, i + 1, i + 2)
  }
```

```
  return this
}

Buffer.prototype.swap64 = function swap64 () {
  var len = this.length
  if (len % 8 !== 0) {
    throw new RangeError('Buffer size must be a multiple of 64-bits')
  }
  for (var i = 0; i < len; i += 8) {
    swap(this, i, i + 7)
    swap(this, i + 1, i + 6)
    swap(this, i + 2, i + 5)
    swap(this, i + 3, i + 4)
  }
  return this
}

Buffer.prototype.toString = function toString () {
  var length = this.length
  if (length === 0) return ''
  if (arguments.length === 0) return utf8Slice(this, 0, length)
  return slowToString.apply(this, arguments)
}
```

```
Buffer.prototype.equals = function equals (b) {
  if (!Buffer.isBuffer(b)) throw new TypeError('Argument must be a Buffer')
  if (this === b) return true
  return Buffer.compare(this, b) === 0
}

Buffer.prototype.inspect = function inspect () {
  var str = ''
  var max = exports.INSPECT_MAX_BYTES
  if (this.length > 0) {
    str = this.toString('hex', 0, max).match(/.{2}/g).join(' ')
    if (this.length > max) str += ' ... '
  }
  return '<Buffer ' + str + '>'
}

Buffer.prototype.compare = function compare (target, start, end, thisStart,
thisEnd) {
  if (!Buffer.isBuffer(target)) {
    throw new TypeError('Argument must be a Buffer')
  }

  if (start === undefined) {
    start = 0
  }
```

```
if (end === undefined) {
  end = target ? target.length : 0
}
if (thisStart === undefined) {
  thisStart = 0
}
if (thisEnd === undefined) {
  thisEnd = this.length
}

if (start < 0 || end > target.length || thisStart < 0 || thisEnd > this.length) {
  throw new RangeError('out of range index')
}

if (thisStart >= thisEnd && start >= end) {
  return 0
}
if (thisStart >= thisEnd) {
  return -1
}
if (start >= end) {
  return 1
}
```

```
start >>>= 0
end >>>= 0
thisStart >>>= 0
thisEnd >>>= 0

if (this === target) return 0

var x = thisEnd - thisStart
var y = end - start
var len = Math.min(x, y)

var thisCopy = this.slice(thisStart, thisEnd)
var targetCopy = target.slice(start, end)

for (var i = 0; i < len; ++i) {
  if (thisCopy[i] !== targetCopy[i]) {
    x = thisCopy[i]
    y = targetCopy[i]
    break
  }
}

if (x < y) return -1
if (y < x) return 1
```

```
  return 0
}

// Finds either the first index of `val` in `buffer` at offset >= `byteOffset`,
// OR the last index of `val` in `buffer` at offset <= `byteOffset`.
//
// Arguments:
// - buffer - a Buffer to search
// - val - a string, Buffer, or number
// - byteOffset - an index into `buffer`; will be clamped to an int32
// - encoding - an optional encoding, relevant is val is a string
// - dir - true for indexOf, false for lastIndexOf
function bidirectionalIndexOf (buffer, val, byteOffset, encoding, dir) {
  // Empty buffer means no match
  if (buffer.length === 0) return -1

  // Normalize byteOffset
  if (typeof byteOffset === 'string') {
    encoding = byteOffset
    byteOffset = 0
  } else if (byteOffset > 0x7fffffff) {
    byteOffset = 0x7fffffff
  } else if (byteOffset < -0x80000000) {
    byteOffset = -0x80000000
```

```
}
byteOffset = +byteOffset  // Coerce to Number.
if (isNaN(byteOffset)) {
  // byteOffset: it it's undefined, null, NaN, "foo", etc, search whole buffer
  byteOffset = dir ? 0 : (buffer.length - 1)
}

// Normalize byteOffset: negative offsets start from the end of the buffer
if (byteOffset < 0) byteOffset = buffer.length + byteOffset
if (byteOffset >= buffer.length) {
  if (dir) return -1
  else byteOffset = buffer.length - 1
} else if (byteOffset < 0) {
  if (dir) byteOffset = 0
  else return -1
}

// Normalize val
if (typeof val === 'string') {
  val = Buffer.from(val, encoding)
}

// Finally, search either indexOf (if dir is true) or lastIndexOf
if (Buffer.isBuffer(val)) {
```

```
    // Special case: looking for empty string/buffer always fails
    if (val.length === 0) {
      return -1
    }
    return arrayIndexOf(buffer, val, byteOffset, encoding, dir)
  } else if (typeof val === 'number') {
    val = val & 0xFF // Search for a byte value [0-255]
    if (typeof Uint8Array.prototype.indexOf === 'function') {
      if (dir) {
        return Uint8Array.prototype.indexOf.call(buffer, val, byteOffset)
      } else {
        return Uint8Array.prototype.lastIndexOf.call(buffer, val, byteOffset)
      }
    }
    return arrayIndexOf(buffer, [ val ], byteOffset, encoding, dir)
  }

  throw new TypeError('val must be string, number or Buffer')
}

function arrayIndexOf (arr, val, byteOffset, encoding, dir) {
  var indexSize = 1
  var arrLength = arr.length
  var valLength = val.length
```

```
if (encoding !== undefined) {
  encoding = String(encoding).toLowerCase()
  if (encoding === 'ucs2' || encoding === 'ucs-2' ||
      encoding === 'utf16le' || encoding === 'utf-16le') {
    if (arr.length < 2 || val.length < 2) {
      return -1
    }
    indexSize = 2
    arrLength /= 2
    valLength /= 2
    byteOffset /= 2
  }
}

function read (buf, i) {
  if (indexSize === 1) {
    return buf[i]
  } else {
    return buf.readUInt16BE(i * indexSize)
  }
}

var i
```

```
if (dir) {
  var foundIndex = -1
  for (i = byteOffset; i < arrLength; i++) {
    if (read(arr, i) === read(val, foundIndex === -1 ? 0 : i - foundIndex)) {
      if (foundIndex === -1) foundIndex = i
      if (i - foundIndex + 1 === valLength) return foundIndex * indexSize
    } else {
      if (foundIndex !== -1) i -= i - foundIndex
      foundIndex = -1
    }
  }
} else {
  if (byteOffset + valLength > arrLength) byteOffset = arrLength - valLength
  for (i = byteOffset; i >= 0; i--) {
    var found = true
    for (var j = 0; j < valLength; j++) {
      if (read(arr, i + j) !== read(val, j)) {
        found = false
        break
      }
    }
    if (found) return i
  }
}
```

```
  return -1
}

Buffer.prototype.includes = function includes (val, byteOffset, encoding) {
  return this.indexOf(val, byteOffset, encoding) !== -1
}

Buffer.prototype.indexOf = function indexOf (val, byteOffset, encoding) {
  return bidirectionalIndexOf(this, val, byteOffset, encoding, true)
}

Buffer.prototype.lastIndexOf = function lastIndexOf (val, byteOffset, encoding) {
  return bidirectionalIndexOf(this, val, byteOffset, encoding, false)
}

function hexWrite (buf, string, offset, length) {
  offset = Number(offset) || 0
  var remaining = buf.length - offset
  if (!length) {
    length = remaining
  } else {
    length = Number(length)
    if (length > remaining) {
```

```
      length = remaining
    }
  }

  // must be an even number of digits
  var strLen = string.length
  if (strLen % 2 !== 0) throw new TypeError('Invalid hex string')

  if (length > strLen / 2) {
    length = strLen / 2
  }
  for (var i = 0; i < length; ++i) {
    var parsed = parseInt(string.substr(i * 2, 2), 16)
    if (isNaN(parsed)) return i
    buf[offset + i] = parsed
  }
  return i
}

function utf8Write (buf, string, offset, length) {
  return blitBuffer(utf8ToBytes(string, buf.length - offset), buf, offset, length)
}

function asciiWrite (buf, string, offset, length) {
```

```
  return blitBuffer(asciiToBytes(string), buf, offset, length)
}

function latin1Write (buf, string, offset, length) {
  return asciiWrite(buf, string, offset, length)
}

function base64Write (buf, string, offset, length) {
  return blitBuffer(base64ToBytes(string), buf, offset, length)
}

function ucs2Write (buf, string, offset, length) {
  return blitBuffer(utf16leToBytes(string, buf.length - offset), buf, offset, length)
}

Buffer.prototype.write = function write (string, offset, length, encoding) {
  // Buffer#write(string)
  if (offset === undefined) {
    encoding = 'utf8'
    length = this.length
    offset = 0
  // Buffer#write(string, encoding)
  } else if (length === undefined && typeof offset === 'string') {
    encoding = offset
```

```
  length = this.length
  offset = 0
// Buffer#write(string, offset[, length][, encoding])
} else if (isFinite(offset)) {
  offset = offset >>> 0
  if (isFinite(length)) {
    length = length >>> 0
    if (encoding === undefined) encoding = 'utf8'
  } else {
    encoding = length
    length = undefined
  }
// legacy write(string, encoding, offset, length) - remove in v0.13
} else {
  throw new Error(
    'Buffer.write(string, encoding, offset[, length]) is no longer supported'
  )
}

var remaining = this.length - offset
if (length === undefined || length > remaining) length = remaining

if ((string.length > 0 && (length < 0 || offset < 0)) || offset > this.length) {
  throw new RangeError('Attempt to write outside buffer bounds')
```

```
}

if (!encoding) encoding = 'utf8'

var loweredCase = false
for (;;) {
  switch (encoding) {
    case 'hex':
      return hexWrite(this, string, offset, length)

    case 'utf8':
    case 'utf-8':
      return utf8Write(this, string, offset, length)

    case 'ascii':
      return asciiWrite(this, string, offset, length)

    case 'latin1':
    case 'binary':
      return latin1Write(this, string, offset, length)

    case 'base64':
      // Warning: maxLength not taken into account in base64Write
      return base64Write(this, string, offset, length)
```

```
      case 'ucs2':
      case 'ucs-2':
      case 'utf16le':
      case 'utf-16le':
        return ucs2Write(this, string, offset, length)

      default:
        if (loweredCase) throw new TypeError('Unknown encoding: ' + encoding)
        encoding = ('' + encoding).toLowerCase()
        loweredCase = true
    }
  }
}

Buffer.prototype.toJSON = function toJSON () {
  return {
    type: 'Buffer',
    data: Array.prototype.slice.call(this._arr || this, 0)
  }
}

function base64Slice (buf, start, end) {
  if (start === 0 && end === buf.length) {
```

```
    return base64.fromByteArray(buf)
  } else {
    return base64.fromByteArray(buf.slice(start, end))
  }
}

function utf8Slice (buf, start, end) {
  end = Math.min(buf.length, end)
  var res = []

  var i = start
  while (i < end) {
    var firstByte = buf[i]
    var codePoint = null
    var bytesPerSequence = (firstByte > 0xEF) ? 4
      : (firstByte > 0xDF) ? 3
      : (firstByte > 0xBF) ? 2
      : 1

    if (i + bytesPerSequence <= end) {
      var secondByte, thirdByte, fourthByte, tempCodePoint

      switch (bytesPerSequence) {
        case 1:
```

```
            if (firstByte < 0x80) {
              codePoint = firstByte
            }
            break
          case 2:
            secondByte = buf[i + 1]
            if ((secondByte & 0xC0) === 0x80) {
              tempCodePoint = (firstByte & 0x1F) << 0x6 | (secondByte & 0x3F)
              if (tempCodePoint > 0x7F) {
                codePoint = tempCodePoint
              }
            }
            break
          case 3:
            secondByte = buf[i + 1]
            thirdByte = buf[i + 2]
            if ((secondByte & 0xC0) === 0x80 && (thirdByte & 0xC0) === 0x80) {
              tempCodePoint = (firstByte & 0xF) << 0xC | (secondByte & 0x3F) << 0x6 |
(thirdByte & 0x3F)
              if (tempCodePoint > 0x7FF && (tempCodePoint < 0xD800 ||
tempCodePoint > 0xDFFF)) {
                codePoint = tempCodePoint
              }
            }
            break
```

```
          case 4:
            secondByte = buf[i + 1]
            thirdByte = buf[i + 2]
            fourthByte = buf[i + 3]
            if ((secondByte & 0xC0) === 0x80 && (thirdByte & 0xC0) === 0x80 &&
(fourthByte & 0xC0) === 0x80) {
              tempCodePoint = (firstByte & 0xF) << 0x12 | (secondByte & 0x3F) << 0xC
| (thirdByte & 0x3F) << 0x6 | (fourthByte & 0x3F)
              if (tempCodePoint > 0xFFFF && tempCodePoint < 0x110000) {
                codePoint = tempCodePoint
              }
            }
        }
      }

      if (codePoint === null) {
        // we did not generate a valid codePoint so insert a
        // replacement char (U+FFFD) and advance only 1 byte
        codePoint = 0xFFFD
        bytesPerSequence = 1
      } else if (codePoint > 0xFFFF) {
        // encode to utf16 (surrogate pair dance)
        codePoint -= 0x10000
        res.push(codePoint >>> 10 & 0x3FF | 0xD800)
        codePoint = 0xDC00 | codePoint & 0x3FF
```

```
    }

    res.push(codePoint)
    i += bytesPerSequence
  }

  return decodeCodePointsArray(res)
}

// Based on http://stackoverflow.com/a/22747272/680742, the browser with
// the lowest limit is Chrome, with 0x10000 args.
// We go 1 magnitude less, for safety
var MAX_ARGUMENTS_LENGTH = 0x1000

function decodeCodePointsArray (codePoints) {
  var len = codePoints.length
  if (len <= MAX_ARGUMENTS_LENGTH) {
    return String.fromCharCode.apply(String, codePoints) // avoid extra slice()
  }

  // Decode in chunks to avoid "call stack size exceeded".
  var res = ''
  var i = 0
  while (i < len) {
```

```
    res += String.fromCharCode.apply(
      String,
      codePoints.slice(i, i += MAX_ARGUMENTS_LENGTH)
    )
  }
  return res
}

function asciiSlice (buf, start, end) {
  var ret = ''
  end = Math.min(buf.length, end)

  for (var i = start; i < end; ++i) {
    ret += String.fromCharCode(buf[i] & 0x7F)
  }
  return ret
}

function latin1Slice (buf, start, end) {
  var ret = ''
  end = Math.min(buf.length, end)

  for (var i = start; i < end; ++i) {
    ret += String.fromCharCode(buf[i])
```

```
  }
  return ret
}

function hexSlice (buf, start, end) {
  var len = buf.length

  if (!start || start < 0) start = 0
  if (!end || end < 0 || end > len) end = len

  var out = ''
  for (var i = start; i < end; ++i) {
    out += toHex(buf[i])
  }
  return out
}

function utf16leSlice (buf, start, end) {
  var bytes = buf.slice(start, end)
  var res = ''
  for (var i = 0; i < bytes.length; i += 2) {
    res += String.fromCharCode(bytes[i] + bytes[i + 1] * 256)
  }
  return res
```

```
}

Buffer.prototype.slice = function slice (start, end) {
  var len = this.length
  start = ~~start
  end = end === undefined ? len : ~~end

  if (start < 0) {
    start += len
    if (start < 0) start = 0
  } else if (start > len) {
    start = len
  }

  if (end < 0) {
    end += len
    if (end < 0) end = 0
  } else if (end > len) {
    end = len
  }

  if (end < start) end = start

  var newBuf = this.subarray(start, end)
```

```
    // Return an augmented `Uint8Array` instance
    newBuf.__proto__ = Buffer.prototype
    return newBuf
  }

  /*
   * Need to make sure that buffer isn't trying to write out of bounds.
   */
  function checkOffset (offset, ext, length) {
    if ((offset % 1) !== 0 || offset < 0) throw new RangeError('offset is not uint')
    if (offset + ext > length) throw new RangeError('Trying to access beyond buffer
length')
  }

  Buffer.prototype.readUIntLE = function readUIntLE (offset, byteLength,
noAssert) {
    offset = offset >>> 0
    byteLength = byteLength >>> 0
    if (!noAssert) checkOffset(offset, byteLength, this.length)

    var val = this[offset]
    var mul = 1
    var i = 0
    while (++i < byteLength && (mul *= 0x100)) {
      val += this[offset + i] * mul
```

```
    }

    return val
  }

  Buffer.prototype.readUIntBE = function readUIntBE (offset, byteLength,
noAssert) {
    offset = offset >>> 0
    byteLength = byteLength >>> 0
    if (!noAssert) {
      checkOffset(offset, byteLength, this.length)
    }

    var val = this[offset + --byteLength]
    var mul = 1
    while (byteLength > 0 && (mul *= 0x100)) {
      val += this[offset + --byteLength] * mul
    }

    return val
  }

  Buffer.prototype.readUInt8 = function readUInt8 (offset, noAssert) {
    offset = offset >>> 0
    if (!noAssert) checkOffset(offset, 1, this.length)
```

```
  return this[offset]
}

Buffer.prototype.readUInt16LE = function readUInt16LE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 2, this.length)
  return this[offset] | (this[offset + 1] << 8)
}

Buffer.prototype.readUInt16BE = function readUInt16BE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 2, this.length)
  return (this[offset] << 8) | this[offset + 1]
}

Buffer.prototype.readUInt32LE = function readUInt32LE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 4, this.length)

  return ((this[offset]) |
      (this[offset + 1] << 8) |
      (this[offset + 2] << 16)) +
      (this[offset + 3] * 0x1000000)
}
```

```
Buffer.prototype.readUInt32BE = function readUInt32BE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 4, this.length)

  return (this[offset] * 0x1000000) +
    ((this[offset + 1] << 16) |
    (this[offset + 2] << 8) |
    this[offset + 3])
}

Buffer.prototype.readIntLE = function readIntLE (offset, byteLength, noAssert) {
  offset = offset >>> 0
  byteLength = byteLength >>> 0
  if (!noAssert) checkOffset(offset, byteLength, this.length)

  var val = this[offset]
  var mul = 1
  var i = 0
  while (++i < byteLength && (mul *= 0x100)) {
    val += this[offset + i] * mul
  }
  mul *= 0x80
```

```
  if (val >= mul) val -= Math.pow(2, 8 * byteLength)

  return val
}

Buffer.prototype.readIntBE = function readIntBE (offset, byteLength, noAssert) {
  offset = offset >>> 0
  byteLength = byteLength >>> 0
  if (!noAssert) checkOffset(offset, byteLength, this.length)

  var i = byteLength
  var mul = 1
  var val = this[offset + --i]
  while (i > 0 && (mul *= 0x100)) {
    val += this[offset + --i] * mul
  }
  mul *= 0x80

  if (val >= mul) val -= Math.pow(2, 8 * byteLength)

  return val
}

Buffer.prototype.readInt8 = function readInt8 (offset, noAssert) {
```

```
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 1, this.length)
  if (!(this[offset] & 0x80)) return (this[offset])
  return ((0xff - this[offset] + 1) * -1)
}

Buffer.prototype.readInt16LE = function readInt16LE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 2, this.length)
  var val = this[offset] | (this[offset + 1] << 8)
  return (val & 0x8000) ? val | 0xFFFF0000 : val
}

Buffer.prototype.readInt16BE = function readInt16BE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 2, this.length)
  var val = this[offset + 1] | (this[offset] << 8)
  return (val & 0x8000) ? val | 0xFFFF0000 : val
}

Buffer.prototype.readInt32LE = function readInt32LE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 4, this.length)
```

```
  return (this[offset]) |
    (this[offset + 1] << 8) |
    (this[offset + 2] << 16) |
    (this[offset + 3] << 24)
}

Buffer.prototype.readInt32BE = function readInt32BE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 4, this.length)

  return (this[offset] << 24) |
    (this[offset + 1] << 16) |
    (this[offset + 2] << 8) |
    (this[offset + 3])
}

Buffer.prototype.readFloatLE = function readFloatLE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 4, this.length)
  return ieee754.read(this, offset, true, 23, 4)
}

Buffer.prototype.readFloatBE = function readFloatBE (offset, noAssert) {
  offset = offset >>> 0
```

```
  if (!noAssert) checkOffset(offset, 4, this.length)
  return ieee754.read(this, offset, false, 23, 4)
}

Buffer.prototype.readDoubleLE = function readDoubleLE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 8, this.length)
  return ieee754.read(this, offset, true, 52, 8)
}

Buffer.prototype.readDoubleBE = function readDoubleBE (offset, noAssert) {
  offset = offset >>> 0
  if (!noAssert) checkOffset(offset, 8, this.length)
  return ieee754.read(this, offset, false, 52, 8)
}

function checkInt (buf, value, offset, ext, max, min) {
  if (!Buffer.isBuffer(buf)) throw new TypeError('"buffer" argument must be a Buffer instance')
  if (value > max || value < min) throw new RangeError('"value" argument is out of bounds')
  if (offset + ext > buf.length) throw new RangeError('Index out of range')
}
```

```
  Buffer.prototype.writeUIntLE = function writeUIntLE (value, offset, byteLength,
noAssert) {
    value = +value
    offset = offset >>> 0
    byteLength = byteLength >>> 0
    if (!noAssert) {
      var maxBytes = Math.pow(2, 8 * byteLength) - 1
      checkInt(this, value, offset, byteLength, maxBytes, 0)
    }

    var mul = 1
    var i = 0
    this[offset] = value & 0xFF
    while (++i < byteLength && (mul *= 0x100)) {
      this[offset + i] = (value / mul) & 0xFF
    }

    return offset + byteLength
  }

  Buffer.prototype.writeUIntBE = function writeUIntBE (value, offset, byteLength,
noAssert) {
    value = +value
    offset = offset >>> 0
    byteLength = byteLength >>> 0
```

```
    if (!noAssert) {
      var maxBytes = Math.pow(2, 8 * byteLength) - 1
      checkInt(this, value, offset, byteLength, maxBytes, 0)
    }

    var i = byteLength - 1
    var mul = 1
    this[offset + i] = value & 0xFF
    while (--i >= 0 && (mul *= 0x100)) {
      this[offset + i] = (value / mul) & 0xFF
    }

    return offset + byteLength
  }

  Buffer.prototype.writeUInt8 = function writeUInt8 (value, offset, noAssert) {
    value = +value
    offset = offset >>> 0
    if (!noAssert) checkInt(this, value, offset, 1, 0xff, 0)
    this[offset] = (value & 0xff)
    return offset + 1
  }

  Buffer.prototype.writeUInt16LE = function writeUInt16LE (value, offset,
noAssert) {
```

```
    value = +value
    offset = offset >>> 0
    if (!noAssert) checkInt(this, value, offset, 2, 0xffff, 0)
    this[offset] = (value & 0xff)
    this[offset + 1] = (value >>> 8)
    return offset + 2
  }

  Buffer.prototype.writeUInt16BE = function writeUInt16BE (value, offset,
noAssert) {
    value = +value
    offset = offset >>> 0
    if (!noAssert) checkInt(this, value, offset, 2, 0xffff, 0)
    this[offset] = (value >>> 8)
    this[offset + 1] = (value & 0xff)
    return offset + 2
  }

  Buffer.prototype.writeUInt32LE = function writeUInt32LE (value, offset,
noAssert) {
    value = +value
    offset = offset >>> 0
    if (!noAssert) checkInt(this, value, offset, 4, 0xffffffff, 0)
    this[offset + 3] = (value >>> 24)
    this[offset + 2] = (value >>> 16)
```

```
    this[offset + 1] = (value >>> 8)
    this[offset] = (value & 0xff)
    return offset + 4
  }

  Buffer.prototype.writeUInt32BE = function writeUInt32BE (value, offset,
noAssert) {
    value = +value
    offset = offset >>> 0
    if (!noAssert) checkInt(this, value, offset, 4, 0xffffffff, 0)
    this[offset] = (value >>> 24)
    this[offset + 1] = (value >>> 16)
    this[offset + 2] = (value >>> 8)
    this[offset + 3] = (value & 0xff)
    return offset + 4
  }

  Buffer.prototype.writeIntLE = function writeIntLE (value, offset, byteLength,
noAssert) {
    value = +value
    offset = offset >>> 0
    if (!noAssert) {
      var limit = Math.pow(2, 8 * byteLength - 1)

      checkInt(this, value, offset, byteLength, limit - 1, -limit)
```

```
    }

    var i = 0
    var mul = 1
    var sub = 0
    this[offset] = value & 0xFF
    while (++i < byteLength && (mul *= 0x100)) {
      if (value < 0 && sub === 0 && this[offset + i - 1] !== 0) {
        sub = 1
      }
      this[offset + i] = ((value / mul) >> 0) - sub & 0xFF
    }

    return offset + byteLength
  }

  Buffer.prototype.writeIntBE = function writeIntBE (value, offset, byteLength, noAssert) {
    value = +value
    offset = offset >>> 0
    if (!noAssert) {
      var limit = Math.pow(2, 8 * byteLength - 1)

      checkInt(this, value, offset, byteLength, limit - 1, -limit)
    }
```

```
  var i = byteLength - 1
  var mul = 1
  var sub = 0
  this[offset + i] = value & 0xFF
  while (--i >= 0 && (mul *= 0x100)) {
    if (value < 0 && sub === 0 && this[offset + i + 1] !== 0) {
      sub = 1
    }
    this[offset + i] = ((value / mul) >> 0) - sub & 0xFF
  }

  return offset + byteLength
}

Buffer.prototype.writeInt8 = function writeInt8 (value, offset, noAssert) {
  value = +value
  offset = offset >>> 0
  if (!noAssert) checkInt(this, value, offset, 1, 0x7f, -0x80)
  if (value < 0) value = 0xff + value + 1
  this[offset] = (value & 0xff)
  return offset + 1
}
```

```
Buffer.prototype.writeInt16LE = function writeInt16LE (value, offset, noAssert) {
  value = +value
  offset = offset >>> 0
  if (!noAssert) checkInt(this, value, offset, 2, 0x7fff, -0x8000)
  this[offset] = (value & 0xff)
  this[offset + 1] = (value >>> 8)
  return offset + 2
}

Buffer.prototype.writeInt16BE = function writeInt16BE (value, offset, noAssert) {
  value = +value
  offset = offset >>> 0
  if (!noAssert) checkInt(this, value, offset, 2, 0x7fff, -0x8000)
  this[offset] = (value >>> 8)
  this[offset + 1] = (value & 0xff)
  return offset + 2
}

Buffer.prototype.writeInt32LE = function writeInt32LE (value, offset, noAssert) {
  value = +value
  offset = offset >>> 0
  if (!noAssert) checkInt(this, value, offset, 4, 0x7fffffff, -0x80000000)
  this[offset] = (value & 0xff)
  this[offset + 1] = (value >>> 8)
```

```
  this[offset + 2] = (value >>> 16)
  this[offset + 3] = (value >>> 24)
  return offset + 4
}

Buffer.prototype.writeInt32BE = function writeInt32BE (value, offset, noAssert) {
  value = +value
  offset = offset >>> 0
  if (!noAssert) checkInt(this, value, offset, 4, 0x7fffffff, -0x80000000)
  if (value < 0) value = 0xffffffff + value + 1
  this[offset] = (value >>> 24)
  this[offset + 1] = (value >>> 16)
  this[offset + 2] = (value >>> 8)
  this[offset + 3] = (value & 0xff)
  return offset + 4
}

function checkIEEE754 (buf, value, offset, ext, max, min) {
  if (offset + ext > buf.length) throw new RangeError('Index out of range')
  if (offset < 0) throw new RangeError('Index out of range')
}

function writeFloat (buf, value, offset, littleEndian, noAssert) {
  value = +value
```

```
  offset = offset >>> 0
  if (!noAssert) {
    checkIEEE754(buf, value, offset, 4, 3.4028234663852886e+38, -
3.4028234663852886e+38)
  }
  ieee754.write(buf, value, offset, littleEndian, 23, 4)
  return offset + 4
}

Buffer.prototype.writeFloatLE = function writeFloatLE (value, offset, noAssert) {
  return writeFloat(this, value, offset, true, noAssert)
}

Buffer.prototype.writeFloatBE = function writeFloatBE (value, offset, noAssert) {
  return writeFloat(this, value, offset, false, noAssert)
}

function writeDouble (buf, value, offset, littleEndian, noAssert) {
  value = +value
  offset = offset >>> 0
  if (!noAssert) {
    checkIEEE754(buf, value, offset, 8, 1.7976931348623157E+308, -
1.7976931348623157E+308)
  }
  ieee754.write(buf, value, offset, littleEndian, 52, 8)
```

```
    return offset + 8
  }

  Buffer.prototype.writeDoubleLE = function writeDoubleLE (value, offset,
noAssert) {
    return writeDouble(this, value, offset, true, noAssert)
  }

  Buffer.prototype.writeDoubleBE = function writeDoubleBE (value, offset,
noAssert) {
    return writeDouble(this, value, offset, false, noAssert)
  }

  // copy(targetBuffer, targetStart=0, sourceStart=0, sourceEnd=buffer.length)
  Buffer.prototype.copy = function copy (target, targetStart, start, end) {
    if (!start) start = 0
    if (!end && end !== 0) end = this.length
    if (targetStart >= target.length) targetStart = target.length
    if (!targetStart) targetStart = 0
    if (end > 0 && end < start) end = start

    // Copy 0 bytes; we're done
    if (end === start) return 0
    if (target.length === 0 || this.length === 0) return 0
```

```
    // Fatal error conditions
    if (targetStart < 0) {
      throw new RangeError('targetStart out of bounds')
    }
    if (start < 0 || start >= this.length) throw new RangeError('sourceStart out of
bounds')
    if (end < 0) throw new RangeError('sourceEnd out of bounds')

    // Are we oob?
    if (end > this.length) end = this.length
    if (target.length - targetStart < end - start) {
      end = target.length - targetStart + start
    }

    var len = end - start
    var i

    if (this === target && start < targetStart && targetStart < end) {
      // descending copy from end
      for (i = len - 1; i >= 0; --i) {
        target[i + targetStart] = this[i + start]
      }
    } else if (len < 1000) {
      // ascending copy from start
      for (i = 0; i < len; ++i) {
```

```
    target[i + targetStart] = this[i + start]
  }
 } else {
  Uint8Array.prototype.set.call(
    target,
    this.subarray(start, start + len),
    targetStart
  )
 }

 return len
}

// Usage:
//    buffer.fill(number[, offset[, end]])
//    buffer.fill(buffer[, offset[, end]])
//    buffer.fill(string[, offset[, end]][, encoding])
Buffer.prototype.fill = function fill (val, start, end, encoding) {
 // Handle string cases:
 if (typeof val === 'string') {
  if (typeof start === 'string') {
    encoding = start
    start = 0
    end = this.length
```

```
  } else if (typeof end === 'string') {
   encoding = end
   end = this.length
  }
  if (val.length === 1) {
   var code = val.charCodeAt(0)
   if (code < 256) {
    val = code
   }
  }
  if (encoding !== undefined && typeof encoding !== 'string') {
   throw new TypeError('encoding must be a string')
  }
  if (typeof encoding === 'string' && !Buffer.isEncoding(encoding)) {
   throw new TypeError('Unknown encoding: ' + encoding)
  }
 } else if (typeof val === 'number') {
  val = val & 255
 }

 // Invalid ranges are not set to a default, so can range check early.
 if (start < 0 || this.length < start || this.length < end) {
  throw new RangeError('Out of range index')
 }
```

```
if (end <= start) {
  return this
}

start = start >>> 0
end = end === undefined ? this.length : end >>> 0

if (!val) val = 0

var i
if (typeof val === 'number') {
  for (i = start; i < end; ++i) {
    this[i] = val
  }
} else {
  var bytes = Buffer.isBuffer(val)
    ? val
    : new Buffer(val, encoding)
  var len = bytes.length
  for (i = 0; i < end - start; ++i) {
    this[i + start] = bytes[i % len]
  }
}
```

```
    return this
  }

  // HELPER FUNCTIONS
  // ================

  var INVALID_BASE64_RE = /[^+/0-9A-Za-z-_]/g

  function base64clean (str) {
    // Node strips out invalid characters like \n and \t from the string, base64-js
does not
    str = stringtrim(str).replace(INVALID_BASE64_RE, '')
    // Node converts strings with length < 2 to ''
    if (str.length < 2) return ''
    // Node allows for non-padded base64 strings (missing trailing ===), base64-js
does not
    while (str.length % 4 !== 0) {
      str = str + '='
    }
    return str
  }

  function stringtrim (str) {
    if (str.trim) return str.trim()
```

```
  return str.replace(/^\s+|\s+$/g, '')
}

function toHex (n) {
  if (n < 16) return '0' + n.toString(16)
  return n.toString(16)
}

function utf8ToBytes (string, units) {
  units = units || Infinity
  var codePoint
  var length = string.length
  var leadSurrogate = null
  var bytes = []

  for (var i = 0; i < length; ++i) {
    codePoint = string.charCodeAt(i)

    // is surrogate component
    if (codePoint > 0xD7FF && codePoint < 0xE000) {
      // last char was a lead
      if (!leadSurrogate) {
        // no lead yet
        if (codePoint > 0xDBFF) {
```

```
    // unexpected trail
    if ((units -= 3) > -1) bytes.push(0xEF, 0xBF, 0xBD)
    continue
  } else if (i + 1 === length) {
    // unpaired lead
    if ((units -= 3) > -1) bytes.push(0xEF, 0xBF, 0xBD)
    continue
  }

  // valid lead
  leadSurrogate = codePoint

  continue
}

// 2 leads in a row
if (codePoint < 0xDC00) {
  if ((units -= 3) > -1) bytes.push(0xEF, 0xBF, 0xBD)
  leadSurrogate = codePoint
  continue
}

// valid surrogate pair
codePoint = (leadSurrogate - 0xD800 << 10 | codePoint - 0xDC00) + 0x10000
```

```
} else if (leadSurrogate) {
  // valid bmp char, but last char was a lead
  if ((units -= 3) > -1) bytes.push(0xEF, 0xBF, 0xBD)
}

leadSurrogate = null

// encode utf8
if (codePoint < 0x80) {
  if ((units -= 1) < 0) break
  bytes.push(codePoint)
} else if (codePoint < 0x800) {
  if ((units -= 2) < 0) break
  bytes.push(
    codePoint >> 0x6 | 0xC0,
    codePoint & 0x3F | 0x80
  )
} else if (codePoint < 0x10000) {
  if ((units -= 3) < 0) break
  bytes.push(
    codePoint >> 0xC | 0xE0,
    codePoint >> 0x6 & 0x3F | 0x80,
    codePoint & 0x3F | 0x80
  )
```

```
    } else if (codePoint < 0x110000) {
      if ((units -= 4) < 0) break
      bytes.push(
        codePoint >> 0x12 | 0xF0,
        codePoint >> 0xC & 0x3F | 0x80,
        codePoint >> 0x6 & 0x3F | 0x80,
        codePoint & 0x3F | 0x80
      )
    } else {
      throw new Error('Invalid code point')
    }
  }

  return bytes
}

function asciiToBytes (str) {
  var byteArray = []
  for (var i = 0; i < str.length; ++i) {
    // Node's code seems to be doing this and not & 0x7F..
    byteArray.push(str.charCodeAt(i) & 0xFF)
  }
  return byteArray
}
```

```
function utf16leToBytes (str, units) {
  var c, hi, lo
  var byteArray = []
  for (var i = 0; i < str.length; ++i) {
    if ((units -= 2) < 0) break

    c = str.charCodeAt(i)
    hi = c >> 8
    lo = c % 256
    byteArray.push(lo)
    byteArray.push(hi)
  }

  return byteArray
}

function base64ToBytes (str) {
  return base64.toByteArray(base64clean(str))
}

function blitBuffer (src, dst, offset, length) {
  for (var i = 0; i < length; ++i) {
    if ((i + offset >= dst.length) || (i >= src.length)) break
```

```
    dst[i + offset] = src[i]
  }
  return i
}

function isnan (val) {
  return val !== val // eslint-disable-line no-self-compare
}

},{"base64-js":3,"ieee754":9}],7:[function(require,module,exports){
(function (Buffer){
'use strict';

/* eslint-disable */

var utils = require('./utils/index.js');
var uint256Coder = utils.uint256Coder;
var coderBoolean = utils.coderBoolean;
var coderFixedBytes = utils.coderFixedBytes;
var coderAddress = utils.coderAddress;
var coderDynamicBytes = utils.coderDynamicBytes;
var coderString = utils.coderString;
var coderArray = utils.coderArray;
var paramTypePart = utils.paramTypePart;
```

```
var getParamCoder = utils.getParamCoder;

function Result() {}

function encodeParams(types, values) {
  if (types.length !== values.length) {
    throw new Error('[ethjs-abi] while encoding params, types/values mismatch,
types length ' + types.length + ' should be ' + values.length);
  }

  var parts = [];

  types.forEach(function (type, index) {
    var coder = getParamCoder(type);
    parts.push({ dynamic: coder.dynamic, value: coder.encode(values[index]) });
  });

  function alignSize(size) {
    return parseInt(32 * Math.ceil(size / 32));
  }

  var staticSize = 0,
      dynamicSize = 0;
  parts.forEach(function (part) {
    if (part.dynamic) {
```

```
    staticSize += 32;
    dynamicSize += alignSize(part.value.length);
  } else {
    staticSize += alignSize(part.value.length);
  }
});

var offset = 0,
    dynamicOffset = staticSize;
var data = new Buffer(staticSize + dynamicSize);

parts.forEach(function (part, index) {
  if (part.dynamic) {
    uint256Coder.encode(dynamicOffset).copy(data, offset);
    offset += 32;

    part.value.copy(data, dynamicOffset);
    dynamicOffset += alignSize(part.value.length);
  } else {
    part.value.copy(data, offset);
    offset += alignSize(part.value.length);
  }
});
```

```
  return '0x' + data.toString('hex');
}

// decode bytecode data from output names and types
function decodeParams(names, types, data) {
  // Names is optional, so shift over all the parameters if not provided
  if (arguments.length < 3) {
    data = types;
    types = names;
    names = [];
  }

  data = utils.hexOrBuffer(data);
  var values = new Result();

  var offset = 0;
  types.forEach(function (type, index) {
    var coder = getParamCoder(type);
    if (coder.dynamic) {
      var dynamicOffset = uint256Coder.decode(data, offset);
      var result = coder.decode(data, dynamicOffset.value.toNumber());
      offset += dynamicOffset.consumed;
    } else {
      var result = coder.decode(data, offset);
```

```
      offset += result.consumed;
    }
    values[index] = result.value;
    if (names[index]) {
      values[names[index]] = result.value;
    }
  });
  return values;
}

// encode method ABI object with values in an array, output bytecode
function encodeMethod(method, values) {
  var signature = method.name + '(' + utils.getKeys(method.inputs,
'type').join(',') + ')';
  var signatureEncoded = '0x' + new Buffer(utils.keccak256(signature),
'hex').slice(0, 4).toString('hex');
  var paramsEncoded = encodeParams(utils.getKeys(method.inputs, 'type'),
values).substring(2);

  return '' + signatureEncoded + paramsEncoded;
}

// decode method data bytecode, from method ABI object
function decodeMethod(method, data) {
  var outputNames = utils.getKeys(method.outputs, 'name', true);
```

```
  var outputTypes = utils.getKeys(method.outputs, 'type');

  return decodeParams(outputNames, outputTypes, utils.hexOrBuffer(data));
}

// decode method data bytecode, from method ABI object
function encodeEvent(eventObject, values) {
  return encodeMethod(eventObject, values);
}

// decode method data bytecode, from method ABI object
function decodeEvent(eventObject, data) {
  var inputNames = utils.getKeys(eventObject.inputs, 'name', true);
  var inputTypes = utils.getKeys(eventObject.inputs, 'type');

  return decodeParams(inputNames, inputTypes, utils.hexOrBuffer(data));
}

module.exports = {
  encodeParams: encodeParams,
  decodeParams: decodeParams,
  encodeMethod: encodeMethod,
  decodeMethod: decodeMethod,
  encodeEvent: encodeEvent,
```

```
  decodeEvent: decodeEvent
};
}).call(this,require("buffer").Buffer)
},{"./utils/index.js":8,"buffer":6}],8:[function(require,module,exports){
(function (Buffer){
'use strict';

var BN = require('bn.js');
var numberToBN = require('number-to-bn');
var keccak256 = require('js-sha3').keccak_256;

// from ethereumjs-util
function stripZeros(aInput) {
  var a = aInput; // eslint-disable-line
  var first = a[0]; // eslint-disable-line
  while (a.length > 0 && first.toString() === '0') {
    a = a.slice(1);
    first = a[0];
  }
  return a;
}

function bnToBuffer(bnInput) {
  var bn = bnInput; // eslint-disable-line
```

```
    var hex = bn.toString(16); // eslint-disable-line
    if (hex.length % 2) {
      hex = '0' + hex;
    }
    return stripZeros(new Buffer(hex, 'hex'));
  }

  function isHexString(value, length) {
    if (typeof value !== 'string' || !value.match(/^0x[0-9A-Fa-f]*$/)) {
      return false;
    }
    if (length && value.length !== 2 + 2 * length) {
      return false;
    }
    return true;
  }

  function hexOrBuffer(valueInput, name) {
    var value = valueInput; // eslint-disable-line
    if (!Buffer.isBuffer(value)) {
      if (!isHexString(value)) {
        var error = new Error(name ? '[ethjs-abi] invalid ' + name : '[ethjs-abi] invalid hex or buffer, must be a prefixed alphanumeric even length hex string');
        error.reason = '[ethjs-abi] invalid hex string, hex must be prefixed and alphanumeric (e.g. 0x023..)';
```

```
      error.value = value;
      throw error;
    }

    value = value.substring(2);
    if (value.length % 2) {
      value = '0' + value;
    }
    value = new Buffer(value, 'hex');
  }

  return value;
}

function hexlify(value) {
  if (typeof value === 'number') {
    return '0x' + bnToBuffer(new BN(value)).toString('hex');
  } else if (value.mod || value.modulo) {
    return '0x' + bnToBuffer(value).toString('hex');
  } else {
    // eslint-disable-line
    return '0x' + hexOrBuffer(value).toString('hex');
  }
}
```

```
// getKeys([{a: 1, b: 2}, {a: 3, b: 4}], 'a') => [1, 3]
function getKeys(params, key, allowEmpty) {
  var result = []; // eslint-disable-line

  if (!Array.isArray(params)) {
    throw new Error('[ethjs-abi] while getting keys, invalid params value ' +
JSON.stringify(params));
  }

  for (var i = 0; i < params.length; i++) {
    // eslint-disable-line
    var value = params[i][key]; // eslint-disable-line
    if (allowEmpty && !value) {
      value = '';
    } else if (typeof value !== 'string') {
      throw new Error('[ethjs-abi] while getKeys found invalid ABI data structure,
type value not string');
    }
    result.push(value);
  }

  return result;
}
```

```
    function coderNumber(size, signed) {
      return {
        encode: function encodeNumber(valueInput) {
          var value = valueInput; // eslint-disable-line

          if (typeof value === 'object' && value.toString && (value.toTwos ||
value.dividedToIntegerBy)) {
            value = value.toString(10).split('.')[0];
          }

          if (typeof value === 'string' || typeof value === 'number') {
            value = String(value).split('.')[0];
          }

          value = numberToBN(value);
          value = value.toTwos(size * 8).maskn(size * 8);
          if (signed) {
            value = value.fromTwos(size * 8).toTwos(256);
          }
          return value.toArrayLike(Buffer, 'be', 32);
        },
        decode: function decodeNumber(data, offset) {
          var junkLength = 32 - size; // eslint-disable-line
          var value = new BN(data.slice(offset + junkLength, offset + 32)); // eslint-
disable-line
```

```
      if (signed) {
        value = value.fromTwos(size * 8);
      } else {
        value = value.maskn(size * 8);
      }
      return {
        consumed: 32,
        value: new BN(value.toString(10))
      };
    }
  };
}
var uint256Coder = coderNumber(32, false);

var coderBoolean = {
  encode: function encodeBoolean(value) {
    return uint256Coder.encode(value ? 1 : 0);
  },
  decode: function decodeBoolean(data, offset) {
    var result = uint256Coder.decode(data, offset); // eslint-disable-line
    return {
      consumed: result.consumed,
      value: !result.value.isZero()
    };
```

```
    }
  };

  function coderFixedBytes(length) {
    return {
      encode: function encodeFixedBytes(valueInput) {
        var value = valueInput; // eslint-disable-line
        value = hexOrBuffer(value);

        if (value.length === 32) {
          return value;
        }

        var result = new Buffer(32); // eslint-disable-line
        result.fill(0);
        value.copy(result);
        return result;
      },
      decode: function decodeFixedBytes(data, offset) {
        if (data.length < offset + 32) {
          throw new Error('[ethjs-abi] while decoding fixed bytes, invalid bytes data
length: ' + length);
        }

        return {
```

```
        consumed: 32,
        value: '0x' + data.slice(offset, offset + length).toString('hex')
      };
    }
  };
}

var coderAddress = {
  encode: function encodeAddress(valueInput) {
    var value = valueInput; // eslint-disable-line
    var result = new Buffer(32); // eslint-disable-line
    if (!isHexString(value, 20)) {
      throw new Error('[ethjs-abi] while encoding address, invalid address value,
not alphanumeric 20 byte hex string');
    }
    value = hexOrBuffer(value);
    result.fill(0);
    value.copy(result, 12);
    return result;
  },
  decode: function decodeAddress(data, offset) {
    if (data.length === 0) {
      return {
        consumed: 32,
        value: '0x'
```

```
      };
    }
    if (data.length < offset + 32) {
      throw new Error('[ethjs-abi] while decoding address data, invalid address
data, invalid byte length ' + data.length);
    }
    return {
      consumed: 32,
      value: '0x' + data.slice(offset + 12, offset + 32).toString('hex')
    };
  }
};

function encodeDynamicBytesHelper(value) {
  var dataLength = parseInt(32 * Math.ceil(value.length / 32)); // eslint-disable-
line
  var padding = new Buffer(dataLength - value.length); // eslint-disable-line
  padding.fill(0);

  return Buffer.concat([uint256Coder.encode(value.length), value, padding]);
}

function decodeDynamicBytesHelper(data, offset) {
  if (data.length < offset + 32) {
```

```
      throw new Error('[ethjs-abi] while decoding dynamic bytes data, invalid bytes length: ' + data.length + ' should be less than ' + (offset + 32));
    }

    var length = uint256Coder.decode(data, offset).value; // eslint-disable-line
    length = length.toNumber();
    if (data.length < offset + 32 + length) {
      throw new Error('[ethjs-abi] while decoding dynamic bytes data, invalid bytes length: ' + data.length + ' should be less than ' + (offset + 32 + length));
    }

    return {
      consumed: parseInt(32 + 32 * Math.ceil(length / 32), 10),
      value: data.slice(offset + 32, offset + 32 + length)
    };
  }

  var coderDynamicBytes = {
    encode: function encodeDynamicBytes(value) {
      return encodeDynamicBytesHelper(hexOrBuffer(value));
    },
    decode: function decodeDynamicBytes(data, offset) {
      var result = decodeDynamicBytesHelper(data, offset); // eslint-disable-line
      result.value = '0x' + result.value.toString('hex');
      return result;
```

```
  },
  dynamic: true
};

var coderString = {
  encode: function encodeString(value) {
    return encodeDynamicBytesHelper(new Buffer(value, 'utf8'));
  },
  decode: function decodeString(data, offset) {
    var result = decodeDynamicBytesHelper(data, offset); // eslint-disable-line
    result.value = result.value.toString('utf8');
    return result;
  },
  dynamic: true
};

function coderArray(coder, lengthInput) {
  return {
    encode: function encodeArray(value) {
      var result = new Buffer(0); // eslint-disable-line
      var length = lengthInput; // eslint-disable-line

      if (!Array.isArray(value)) {
        throw new Error('[ethjs-abi] while encoding array, invalid array data, not
type Object (Array)');
```

```
    }

    if (length === -1) {
      length = value.length;
      result = uint256Coder.encode(length);
    }

    if (length !== value.length) {
      throw new Error('[ethjs-abi] while encoding array, size mismatch array
length ' + length + ' does not equal ' + value.length);
    }

    value.forEach(function (resultValue) {
      result = Buffer.concat([result, coder.encode(resultValue)]);
    });

    return result;
  },
  decode: function decodeArray(data, offsetInput) {
    var length = lengthInput; // eslint-disable-line
    var offset = offsetInput; // eslint-disable-line
    // @TODO:
    // if (data.length < offset + length * 32) { throw new Error('invalid array'); }

    var consumed = 0; // eslint-disable-line
```

```
  var decodeResult; // eslint-disable-line

  if (length === -1) {
    decodeResult = uint256Coder.decode(data, offset);
    length = decodeResult.value.toNumber();
    consumed += decodeResult.consumed;
    offset += decodeResult.consumed;
  }

  var value = []; // eslint-disable-line

  for (var i = 0; i < length; i++) {
    // eslint-disable-line
    var loopResult = coder.decode(data, offset);
    consumed += loopResult.consumed;
    offset += loopResult.consumed;
    value.push(loopResult.value);
  }

  return {
    consumed: consumed,
    value: value
  };
},
```

```
      dynamic: lengthInput === -1
    };
  }

  // Break the type up into [staticType][staticArray]*[dynamicArray]? |
[dynamicType] and
  // build the coder up from its parts
  var paramTypePart = new RegExp(/^((u?int|bytes)([0-
9]*)|(address|bool|string)|(\[([0-9]*)\]))/);

  function getParamCoder(typeInput) {
    var type = typeInput; // eslint-disable-line
    var coder = null; // eslint-disable-line
    var invalidTypeErrorMessage = '[ethjs-abi] while getting param coder
(getParamCoder) type value ' + JSON.stringify(type) + ' is either invalid or
unsupported by ethjs-abi.';

    while (type) {
      var part = type.match(paramTypePart); // eslint-disable-line
      if (!part) {
        throw new Error(invalidTypeErrorMessage);
      }
      type = type.substring(part[0].length);

      var prefix = part[2] || part[4] || part[5]; // eslint-disable-line
      switch (prefix) {
```

```
    case 'int':case 'uint':
      if (coder) {
        throw new Error(invalidTypeErrorMessage);
      }
      var intSize = parseInt(part[3] || 256); // eslint-disable-line
      if (intSize === 0 || intSize > 256 || intSize % 8 !== 0) {
        throw new Error('[ethjs-abi] while getting param coder for type ' + type + ', invalid ' + prefix + '<N> width: ' + type);
      }

      coder = coderNumber(intSize / 8, prefix === 'int');
      break;

    case 'bool':
      if (coder) {
        throw new Error(invalidTypeErrorMessage);
      }
      coder = coderBoolean;
      break;

    case 'string':
      if (coder) {
        throw new Error(invalidTypeErrorMessage);
      }
      coder = coderString;
```

```
      break;

    case 'bytes':
      if (coder) {
        throw new Error(invalidTypeErrorMessage);
      }
      if (part[3]) {
        var size = parseInt(part[3]); // eslint-disable-line
        if (size === 0 || size > 32) {
          throw new Error('[ethjs-abi] while getting param coder for prefix bytes,
invalid type ' + type + ', size ' + size + ' should be 0 or greater than 32');
        }
        coder = coderFixedBytes(size);
      } else {
        coder = coderDynamicBytes;
      }
      break;

    case 'address':
      if (coder) {
        throw new Error(invalidTypeErrorMessage);
      }
      coder = coderAddress;
      break;
```

```
      case '[]':
        if (!coder || coder.dynamic) {
          throw new Error(invalidTypeErrorMessage);
        }
        coder = coderArray(coder, -1);
        break;

      // "[0-9+]"
      default:
        if (!coder || coder.dynamic) {
          throw new Error(invalidTypeErrorMessage);
        }
        var defaultSize = parseInt(part[6]); // eslint-disable-line
        coder = coderArray(coder, defaultSize);
    }
  }

  if (!coder) {
    throw new Error(invalidTypeErrorMessage);
  }
  return coder;
}

module.exports = {
```

```
BN: BN,
bnToBuffer: bnToBuffer,
isHexString: isHexString,
hexOrBuffer: hexOrBuffer,
hexlify: hexlify,
stripZeros: stripZeros,

keccak256: keccak256,


getKeys: getKeys,
numberToBN: numberToBN,
coderNumber: coderNumber,
uint256Coder: uint256Coder,
coderBoolean: coderBoolean,
coderFixedBytes: coderFixedBytes,
coderAddress: coderAddress,
coderDynamicBytes: coderDynamicBytes,
coderString: coderString,
coderArray: coderArray,
paramTypePart: paramTypePart,
getParamCoder: getParamCoder
};
}).call(this,require("buffer").Buffer)
},{"bn.js":4,"buffer":6,"js-sha3":11,"number-to-
bn":12}],9:[function(require,module,exports){
```

```
exports.read = function (buffer, offset, isLE, mLen, nBytes) {
 var e, m
 var eLen = nBytes * 8 - mLen - 1
 var eMax = (1 << eLen) - 1
 var eBias = eMax >> 1
 var nBits = -7
 var i = isLE ? (nBytes - 1) : 0
 var d = isLE ? -1 : 1
 var s = buffer[offset + i]

 i += d

 e = s & ((1 << (-nBits)) - 1)
 s >>= (-nBits)
 nBits += eLen
 for (; nBits > 0; e = e * 256 + buffer[offset + i], i += d, nBits -= 8) {}

 m = e & ((1 << (-nBits)) - 1)
 e >>= (-nBits)
 nBits += mLen
 for (; nBits > 0; m = m * 256 + buffer[offset + i], i += d, nBits -= 8) {}

 if (e === 0) {
  e = 1 - eBias
```

```
  } else if (e === eMax) {
    return m ? NaN : ((s ? -1 : 1) * Infinity)
  } else {
    m = m + Math.pow(2, mLen)
    e = e - eBias
  }
  return (s ? -1 : 1) * m * Math.pow(2, e - mLen)
}

exports.write = function (buffer, value, offset, isLE, mLen, nBytes) {
  var e, m, c
  var eLen = nBytes * 8 - mLen - 1
  var eMax = (1 << eLen) - 1
  var eBias = eMax >> 1
  var rt = (mLen === 23 ? Math.pow(2, -24) - Math.pow(2, -77) : 0)
  var i = isLE ? 0 : (nBytes - 1)
  var d = isLE ? 1 : -1
  var s = value < 0 || (value === 0 && 1 / value < 0) ? 1 : 0

  value = Math.abs(value)

  if (isNaN(value) || value === Infinity) {
    m = isNaN(value) ? 1 : 0
    e = eMax
```

```
} else {
  e = Math.floor(Math.log(value) / Math.LN2)
  if (value * (c = Math.pow(2, -e)) < 1) {
    e--
    c *= 2
  }
  if (e + eBias >= 1) {
    value += rt / c
  } else {
    value += rt * Math.pow(2, 1 - eBias)
  }
  if (value * c >= 2) {
    e++
    c /= 2
  }

  if (e + eBias >= eMax) {
    m = 0
    e = eMax
  } else if (e + eBias >= 1) {
    m = (value * c - 1) * Math.pow(2, mLen)
    e = e + eBias
  } else {
    m = value * Math.pow(2, eBias - 1) * Math.pow(2, mLen)
```

```
        e = 0
      }
    }

    for (; mLen >= 8; buffer[offset + i] = m & 0xff, i += d, m /= 256, mLen -= 8) {}

    e = (e << mLen) | m
    eLen += mLen
    for (; eLen > 0; buffer[offset + i] = e & 0xff, i += d, e /= 256, eLen -= 8) {}

    buffer[offset + i - d] |= s * 128
  }

  },{}],10:[function(require,module,exports){
  /**
   * Returns a `Boolean` on whether or not the a `String` starts with '0x'
   * @param {String} str the string input value
   * @return {Boolean} a boolean if it is or is not hex prefixed
   * @throws if the str input is not a string
   */
  module.exports = function isHexPrefixed(str) {
    if (typeof str !== 'string') {
      throw new Error("[is-hex-prefixed] value must be type 'string', is currently
type " + (typeof str) + ", while checking isHexPrefixed.");
    }
```

```
    return str.slice(0, 2) === '0x';
  }

  },{}],11:[function(require,module,exports){
  (function (process,global){
  /**
   * [js-sha3]{@link https://github.com/emn178/js-sha3}
   *
   * @version 0.5.5
   * @author Chen, Yi-Cyuan [emn178@gmail.com]
   * @copyright Chen, Yi-Cyuan 2015-2016
   * @license MIT
   */
  (function (root) {
    'use strict';

    var NODE_JS = typeof process == 'object' && process.versions &&
process.versions.node;
    if (NODE_JS) {
      root = global;
    }
    var COMMON_JS = !root.JS_SHA3_TEST && typeof module == 'object' &&
module.exports;
    var HEX_CHARS = '0123456789abcdef'.split('');
```

```
var SHAKE_PADDING = [31, 7936, 2031616, 520093696];
var KECCAK_PADDING = [1, 256, 65536, 16777216];
var PADDING = [6, 1536, 393216, 100663296];
var SHIFT = [0, 8, 16, 24];
var RC = [1, 0, 32898, 0, 32906, 2147483648, 2147516416, 2147483648, 32907, 0, 2147483649,
        0, 2147516545, 2147483648, 32777, 2147483648, 138, 0, 136, 0, 2147516425, 0,
        2147483658, 0, 2147516555, 0, 139, 2147483648, 32905, 2147483648, 32771,
        2147483648, 32770, 2147483648, 128, 2147483648, 32778, 0, 2147483658, 2147483648,
        2147516545, 2147483648, 32896, 2147483648, 2147483649, 0, 2147516424, 2147483648];
var BITS = [224, 256, 384, 512];
var SHAKE_BITS = [128, 256];
var OUTPUT_TYPES = ['hex', 'buffer', 'arrayBuffer', 'array'];

var createOutputMethod = function (bits, padding, outputType) {
  return function (message) {
    return new Keccak(bits, padding, bits).update(message)[outputType]();
  }
};

var createShakeOutputMethod = function (bits, padding, outputType) {
  return function (message, outputBits) {
```

```
        return new Keccak(bits, padding,
outputBits).update(message)[outputType]();
      }
    };

    var createMethod = function (bits, padding) {
      var method = createOutputMethod(bits, padding, 'hex');
      method.create = function () {
        return new Keccak(bits, padding, bits);
      };
      method.update = function (message) {
        return method.create().update(message);
      };
      for (var i = 0;i < OUTPUT_TYPES.length;++i) {
        var type = OUTPUT_TYPES[i];
        method[type] = createOutputMethod(bits, padding, type);
      }
      return method;
    };

    var createShakeMethod = function (bits, padding) {
      var method = createShakeOutputMethod(bits, padding, 'hex');
      method.create = function (outputBits) {
        return new Keccak(bits, padding, outputBits);
      };
```

```
    method.update = function (message, outputBits) {
      return method.create(outputBits).update(message);
    };
    for (var i = 0;i < OUTPUT_TYPES.length;++i) {
      var type = OUTPUT_TYPES[i];
      method[type] = createShakeOutputMethod(bits, padding, type);
    }
    return method;
  };

  var algorithms = [
    {name: 'keccak', padding: KECCAK_PADDING, bits: BITS, createMethod:
createMethod},
    {name: 'sha3', padding: PADDING, bits: BITS, createMethod: createMethod},
    {name: 'shake', padding: SHAKE_PADDING, bits: SHAKE_BITS, createMethod:
createShakeMethod}
  ];

  var methods = {};

  for (var i = 0;i < algorithms.length;++i) {
    var algorithm = algorithms[i];
    var bits  = algorithm.bits;
    for (var j = 0;j < bits.length;++j) {
```

```
    methods[algorithm.name +'_' + bits[j]] = algorithm.createMethod(bits[j],
algorithm.padding);
   }
  }

  function Keccak(bits, padding, outputBits) {
   this.blocks = [];
   this.s = [];
   this.padding = padding;
   this.outputBits = outputBits;
   this.reset = true;
   this.block = 0;
   this.start = 0;
   this.blockCount = (1600 - (bits << 1)) >> 5;
   this.byteCount = this.blockCount << 2;
   this.outputBlocks = outputBits >> 5;
   this.extraBytes = (outputBits & 31) >> 3;

   for (var i = 0;i < 50;++i) {
    this.s[i] = 0;
   }
  };

  Keccak.prototype.update = function (message) {
   var notString = typeof message != 'string';
```

```
if (notString && message.constructor == root.ArrayBuffer) {
  message = new Uint8Array(message);
}
var length = message.length, blocks = this.blocks, byteCount = this.byteCount,
    blockCount = this.blockCount, index = 0, s = this.s, i, code;

while (index < length) {
  if (this.reset) {
    this.reset = false;
    blocks[0] = this.block;
    for (i = 1;i < blockCount + 1;++i) {
      blocks[i] = 0;
    }
  }
  if (notString) {
    for (i = this.start;index < length && i < byteCount;++index) {
      blocks[i >> 2] |= message[index] << SHIFT[i++ & 3];
    }
  } else {
    for (i = this.start;index < length && i < byteCount;++index) {
      code = message.charCodeAt(index);
      if (code < 0x80) {
        blocks[i >> 2] |= code << SHIFT[i++ & 3];
      } else if (code < 0x800) {
```

```
            blocks[i >> 2] |= (0xc0 | (code >> 6)) << SHIFT[i++ & 3];
            blocks[i >> 2] |= (0x80 | (code & 0x3f)) << SHIFT[i++ & 3];
          } else if (code < 0xd800 || code >= 0xe000) {
            blocks[i >> 2] |= (0xe0 | (code >> 12)) << SHIFT[i++ & 3];
            blocks[i >> 2] |= (0x80 | ((code >> 6) & 0x3f)) << SHIFT[i++ & 3];
            blocks[i >> 2] |= (0x80 | (code & 0x3f)) << SHIFT[i++ & 3];
          } else {
            code = 0x10000 + (((code & 0x3ff) << 10) |
(message.charCodeAt(++index) & 0x3ff));
            blocks[i >> 2] |= (0xf0 | (code >> 18)) << SHIFT[i++ & 3];
            blocks[i >> 2] |= (0x80 | ((code >> 12) & 0x3f)) << SHIFT[i++ & 3];
            blocks[i >> 2] |= (0x80 | ((code >> 6) & 0x3f)) << SHIFT[i++ & 3];
            blocks[i >> 2] |= (0x80 | (code & 0x3f)) << SHIFT[i++ & 3];
          }
        }
      }
      this.lastByteIndex = i;
      if (i >= byteCount) {
        this.start = i - byteCount;
        this.block = blocks[blockCount];
        for (i = 0;i < blockCount;++i) {
          s[i] ^= blocks[i];
        }
        f(s);
        this.reset = true;
```

```
    } else {
      this.start = i;
    }
  }
  return this;
};

Keccak.prototype.finalize = function () {
  var blocks = this.blocks, i = this.lastByteIndex, blockCount = this.blockCount, s
= this.s;
  blocks[i >> 2] |= this.padding[i & 3];
  if (this.lastByteIndex == this.byteCount) {
    blocks[0] = blocks[blockCount];
    for (i = 1;i < blockCount + 1;++i) {
      blocks[i] = 0;
    }
  }
  blocks[blockCount - 1] |= 0x80000000;
  for (i = 0;i < blockCount;++i) {
    s[i] ^= blocks[i];
  }
  f(s);
};

Keccak.prototype.toString = Keccak.prototype.hex = function () {
```

```
this.finalize();

var blockCount = this.blockCount, s = this.s, outputBlocks = this.outputBlocks,
  extraBytes = this.extraBytes, i = 0, j = 0;
var hex = '', block;
while (j < outputBlocks) {
 for (i = 0;i < blockCount && j < outputBlocks;++i, ++j) {
  block = s[i];
  hex += HEX_CHARS[(block >> 4) & 0x0F] + HEX_CHARS[block & 0x0F] +
      HEX_CHARS[(block >> 12) & 0x0F] + HEX_CHARS[(block >> 8) & 0x0F] +
      HEX_CHARS[(block >> 20) & 0x0F] + HEX_CHARS[(block >> 16) & 0x0F]
+
      HEX_CHARS[(block >> 28) & 0x0F] + HEX_CHARS[(block >> 24) & 0x0F];
 }
 if (j % blockCount == 0) {
  f(s);
  i = 0;
 }
}
if (extraBytes) {
 block = s[i];
 if (extraBytes > 0) {
  hex += HEX_CHARS[(block >> 4) & 0x0F] + HEX_CHARS[block & 0x0F];
 }
 if (extraBytes > 1) {
```

```
          hex += HEX_CHARS[(block >> 12) & 0x0F] + HEX_CHARS[(block >> 8) &
0x0F];
        }
        if (extraBytes > 2) {
          hex += HEX_CHARS[(block >> 20) & 0x0F] + HEX_CHARS[(block >> 16) &
0x0F];
        }
      }
      return hex;
    };

    Keccak.prototype.arrayBuffer = function () {
      this.finalize();

      var blockCount = this.blockCount, s = this.s, outputBlocks = this.outputBlocks,
          extraBytes = this.extraBytes, i = 0, j = 0;
      var bytes = this.outputBits >> 3;
      var buffer;
      if (extraBytes) {
        buffer = new ArrayBuffer((outputBlocks + 1) << 2);
      } else {
        buffer = new ArrayBuffer(bytes);
      }
      var array = new Uint32Array(buffer);
      while (j < outputBlocks) {
```

```
    for (i = 0;i < blockCount && j < outputBlocks;++i, ++j) {
      array[j] = s[i];
    }
    if (j % blockCount == 0) {
      f(s);
    }
  }
  if (extraBytes) {
    array[i] = s[i];
    buffer = buffer.slice(0, bytes);
  }
  return buffer;
};

Keccak.prototype.buffer = Keccak.prototype.arrayBuffer;

Keccak.prototype.digest = Keccak.prototype.array = function () {
  this.finalize();

  var blockCount = this.blockCount, s = this.s, outputBlocks = this.outputBlocks,
      extraBytes = this.extraBytes, i = 0, j = 0;
  var array = [], offset, block;
  while (j < outputBlocks) {
    for (i = 0;i < blockCount && j < outputBlocks;++i, ++j) {
```

```
      offset = j << 2;
      block = s[i];
      array[offset] = block & 0xFF;
      array[offset + 1] = (block >> 8) & 0xFF;
      array[offset + 2] = (block >> 16) & 0xFF;
      array[offset + 3] = (block >> 24) & 0xFF;
    }
    if (j % blockCount == 0) {
      f(s);
    }
  }
  if (extraBytes) {
    offset = j << 2;
    block = s[i];
    if (extraBytes > 0) {
      array[offset] = block & 0xFF;
    }
    if (extraBytes > 1) {
      array[offset + 1] = (block >> 8) & 0xFF;
    }
    if (extraBytes > 2) {
      array[offset + 2] = (block >> 16) & 0xFF;
    }
  }
```

```
    return array;
  };

  var f = function (s) {
    var h, l, n, c0, c1, c2, c3, c4, c5, c6, c7, c8, c9,
        b0, b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12, b13, b14, b15, b16,
b17,
        b18, b19, b20, b21, b22, b23, b24, b25, b26, b27, b28, b29, b30, b31, b32,
b33,
        b34, b35, b36, b37, b38, b39, b40, b41, b42, b43, b44, b45, b46, b47, b48,
b49;
    for (n = 0;n < 48;n += 2) {
      c0 = s[0] ^ s[10] ^ s[20] ^ s[30] ^ s[40];
      c1 = s[1] ^ s[11] ^ s[21] ^ s[31] ^ s[41];
      c2 = s[2] ^ s[12] ^ s[22] ^ s[32] ^ s[42];
      c3 = s[3] ^ s[13] ^ s[23] ^ s[33] ^ s[43];
      c4 = s[4] ^ s[14] ^ s[24] ^ s[34] ^ s[44];
      c5 = s[5] ^ s[15] ^ s[25] ^ s[35] ^ s[45];
      c6 = s[6] ^ s[16] ^ s[26] ^ s[36] ^ s[46];
      c7 = s[7] ^ s[17] ^ s[27] ^ s[37] ^ s[47];
      c8 = s[8] ^ s[18] ^ s[28] ^ s[38] ^ s[48];
      c9 = s[9] ^ s[19] ^ s[29] ^ s[39] ^ s[49];

      h = c8 ^ ((c2 << 1) | (c3 >>> 31));
      l = c9 ^ ((c3 << 1) | (c2 >>> 31));
```

```
s[0] ^= h;
s[1] ^= l;
s[10] ^= h;
s[11] ^= l;
s[20] ^= h;
s[21] ^= l;
s[30] ^= h;
s[31] ^= l;
s[40] ^= h;
s[41] ^= l;
h = c0 ^ ((c4 << 1) | (c5 >>> 31));
l = c1 ^ ((c5 << 1) | (c4 >>> 31));
s[2] ^= h;
s[3] ^= l;
s[12] ^= h;
s[13] ^= l;
s[22] ^= h;
s[23] ^= l;
s[32] ^= h;
s[33] ^= l;
s[42] ^= h;
s[43] ^= l;
h = c2 ^ ((c6 << 1) | (c7 >>> 31));
l = c3 ^ ((c7 << 1) | (c6 >>> 31));
```

```
s[4] ^= h;
s[5] ^= l;
s[14] ^= h;
s[15] ^= l;
s[24] ^= h;
s[25] ^= l;
s[34] ^= h;
s[35] ^= l;
s[44] ^= h;
s[45] ^= l;
h = c4 ^ ((c8 << 1) | (c9 >>> 31));
l = c5 ^ ((c9 << 1) | (c8 >>> 31));
s[6] ^= h;
s[7] ^= l;
s[16] ^= h;
s[17] ^= l;
s[26] ^= h;
s[27] ^= l;
s[36] ^= h;
s[37] ^= l;
s[46] ^= h;
s[47] ^= l;
h = c6 ^ ((c0 << 1) | (c1 >>> 31));
l = c7 ^ ((c1 << 1) | (c0 >>> 31));
```

```
s[8] ^= h;
s[9] ^= l;
s[18] ^= h;
s[19] ^= l;
s[28] ^= h;
s[29] ^= l;
s[38] ^= h;
s[39] ^= l;
s[48] ^= h;
s[49] ^= l;

b0 = s[0];
b1 = s[1];
b32 = (s[11] << 4) | (s[10] >>> 28);
b33 = (s[10] << 4) | (s[11] >>> 28);
b14 = (s[20] << 3) | (s[21] >>> 29);
b15 = (s[21] << 3) | (s[20] >>> 29);
b46 = (s[31] << 9) | (s[30] >>> 23);
b47 = (s[30] << 9) | (s[31] >>> 23);
b28 = (s[40] << 18) | (s[41] >>> 14);
b29 = (s[41] << 18) | (s[40] >>> 14);
b20 = (s[2] << 1) | (s[3] >>> 31);
b21 = (s[3] << 1) | (s[2] >>> 31);
b2 = (s[13] << 12) | (s[12] >>> 20);
```

```
b3 = (s[12] << 12) | (s[13] >>> 20);
b34 = (s[22] << 10) | (s[23] >>> 22);
b35 = (s[23] << 10) | (s[22] >>> 22);
b16 = (s[33] << 13) | (s[32] >>> 19);
b17 = (s[32] << 13) | (s[33] >>> 19);
b48 = (s[42] << 2) | (s[43] >>> 30);
b49 = (s[43] << 2) | (s[42] >>> 30);
b40 = (s[5] << 30) | (s[4] >>> 2);
b41 = (s[4] << 30) | (s[5] >>> 2);
b22 = (s[14] << 6) | (s[15] >>> 26);
b23 = (s[15] << 6) | (s[14] >>> 26);
b4 = (s[25] << 11) | (s[24] >>> 21);
b5 = (s[24] << 11) | (s[25] >>> 21);
b36 = (s[34] << 15) | (s[35] >>> 17);
b37 = (s[35] << 15) | (s[34] >>> 17);
b18 = (s[45] << 29) | (s[44] >>> 3);
b19 = (s[44] << 29) | (s[45] >>> 3);
b10 = (s[6] << 28) | (s[7] >>> 4);
b11 = (s[7] << 28) | (s[6] >>> 4);
b42 = (s[17] << 23) | (s[16] >>> 9);
b43 = (s[16] << 23) | (s[17] >>> 9);
b24 = (s[26] << 25) | (s[27] >>> 7);
b25 = (s[27] << 25) | (s[26] >>> 7);
b6 = (s[36] << 21) | (s[37] >>> 11);
```

```
b7 = (s[37] << 21) | (s[36] >>> 11);
b38 = (s[47] << 24) | (s[46] >>> 8);
b39 = (s[46] << 24) | (s[47] >>> 8);
b30 = (s[8] << 27) | (s[9] >>> 5);
b31 = (s[9] << 27) | (s[8] >>> 5);
b12 = (s[18] << 20) | (s[19] >>> 12);
b13 = (s[19] << 20) | (s[18] >>> 12);
b44 = (s[29] << 7) | (s[28] >>> 25);
b45 = (s[28] << 7) | (s[29] >>> 25);
b26 = (s[38] << 8) | (s[39] >>> 24);
b27 = (s[39] << 8) | (s[38] >>> 24);
b8 = (s[48] << 14) | (s[49] >>> 18);
b9 = (s[49] << 14) | (s[48] >>> 18);

s[0] = b0 ^ (~b2 & b4);
s[1] = b1 ^ (~b3 & b5);
s[10] = b10 ^ (~b12 & b14);
s[11] = b11 ^ (~b13 & b15);
s[20] = b20 ^ (~b22 & b24);
s[21] = b21 ^ (~b23 & b25);
s[30] = b30 ^ (~b32 & b34);
s[31] = b31 ^ (~b33 & b35);
s[40] = b40 ^ (~b42 & b44);
s[41] = b41 ^ (~b43 & b45);
```

```
s[2] = b2 ^ (~b4 & b6);

s[3] = b3 ^ (~b5 & b7);

s[12] = b12 ^ (~b14 & b16);

s[13] = b13 ^ (~b15 & b17);

s[22] = b22 ^ (~b24 & b26);

s[23] = b23 ^ (~b25 & b27);

s[32] = b32 ^ (~b34 & b36);

s[33] = b33 ^ (~b35 & b37);

s[42] = b42 ^ (~b44 & b46);

s[43] = b43 ^ (~b45 & b47);

s[4] = b4 ^ (~b6 & b8);

s[5] = b5 ^ (~b7 & b9);

s[14] = b14 ^ (~b16 & b18);

s[15] = b15 ^ (~b17 & b19);

s[24] = b24 ^ (~b26 & b28);

s[25] = b25 ^ (~b27 & b29);

s[34] = b34 ^ (~b36 & b38);

s[35] = b35 ^ (~b37 & b39);

s[44] = b44 ^ (~b46 & b48);

s[45] = b45 ^ (~b47 & b49);

s[6] = b6 ^ (~b8 & b0);

s[7] = b7 ^ (~b9 & b1);

s[16] = b16 ^ (~b18 & b10);

s[17] = b17 ^ (~b19 & b11);
```

```
    s[26] = b26 ^ (~b28 & b20);
    s[27] = b27 ^ (~b29 & b21);
    s[36] = b36 ^ (~b38 & b30);
    s[37] = b37 ^ (~b39 & b31);
    s[46] = b46 ^ (~b48 & b40);
    s[47] = b47 ^ (~b49 & b41);
    s[8] = b8 ^ (~b0 & b2);
    s[9] = b9 ^ (~b1 & b3);
    s[18] = b18 ^ (~b10 & b12);
    s[19] = b19 ^ (~b11 & b13);
    s[28] = b28 ^ (~b20 & b22);
    s[29] = b29 ^ (~b21 & b23);
    s[38] = b38 ^ (~b30 & b32);
    s[39] = b39 ^ (~b31 & b33);
    s[48] = b48 ^ (~b40 & b42);
    s[49] = b49 ^ (~b41 & b43);

    s[0] ^= RC[n];
    s[1] ^= RC[n + 1];
  }
}

if (COMMON_JS) {
  module.exports = methods;
```

```
      } else if (root) {
        for (var key in methods) {
          root[key] = methods[key];
        }
      }
    }(this));

    }).call(this,require('_process'),typeof global !== "undefined" ? global : typeof self !== "undefined" ? self : typeof window !== "undefined" ? window : {})
    },{"_process":13}],12:[function(require,module,exports){
    var BN = require('bn.js');
    var stripHexPrefix = require('strip-hex-prefix');

    /**
     * Returns a BN object, converts a number value to a BN
     * @param {String|Number|Object} `arg` input a string number, hex string number, number, BigNumber or BN object
     * @return {Object} `output` BN object of the number
     * @throws if the argument is not an array, object that isn't a bignumber, not a string number or number
     */
    module.exports = function numberToBN(arg) {
      if (typeof arg === 'string' || typeof arg === 'number') {
        var multiplier = new BN(1); // eslint-disable-line
        var formattedString = String(arg).toLowerCase().trim();
```

```
    var isHexPrefixed = formattedString.substr(0, 2) === '0x' ||
formattedString.substr(0, 3) === '-0x';

    var stringArg = stripHexPrefix(formattedString); // eslint-disable-line

    if (stringArg.substr(0, 1) === '-') {

      stringArg = stripHexPrefix(stringArg.slice(1));

      multiplier = new BN(-1, 10);

    }

    stringArg = stringArg === '' ? '0' : stringArg;


    if ((!stringArg.match(/^-?[0-9]+$/) && stringArg.match(/^[0-9A-Fa-f]+$/))

      || stringArg.match(/^[a-fA-F]+$/)

      || (isHexPrefixed === true && stringArg.match(/^[0-9A-Fa-f]+$/))) {

      return new BN(stringArg, 16).mul(multiplier);

    }


    if ((stringArg.match(/^-?[0-9]+$/) || stringArg === '') && isHexPrefixed ===
false) {

      return new BN(stringArg, 10).mul(multiplier);

    }

  } else if (typeof arg === 'object' && arg.toString && (!arg.pop && !arg.push)) {

    if (arg.toString(10).match(/^-?[0-9]+$/) && (arg.mul ||
arg.dividedToIntegerBy)) {

      return new BN(arg.toString(10), 10);

    }

  }
```

```
    throw new Error('[number-to-bn] while converting number ' +
JSON.stringify(arg) + ' to BN.js instance, error: invalid number value. Value must be an integer, hex string, BN or BigNumber instance. Note, decimals are not supported.');
  }

  },{"bn.js":4,"strip-hex-prefix":14}],13:[function(require,module,exports){
  // shim for using process in browser
  var process = module.exports = {};

  // cached from whatever global is present so that test runners that stub it
  // don't break things.  But we need to wrap it in a try catch in case it is
  // wrapped in strict mode code which doesn't define any globals.  It's inside a
  // function because try/catches deoptimize in certain engines.

  var cachedSetTimeout;
  var cachedClearTimeout;

  function defaultSetTimout() {
    throw new Error('setTimeout has not been defined');
  }
  function defaultClearTimeout () {
    throw new Error('clearTimeout has not been defined');
  }
```

```
(function () {
    try {
        if (typeof setTimeout === 'function') {
            cachedSetTimeout = setTimeout;
        } else {
            cachedSetTimeout = defaultSetTimout;
        }
    } catch (e) {
        cachedSetTimeout = defaultSetTimout;
    }
    try {
        if (typeof clearTimeout === 'function') {
            cachedClearTimeout = clearTimeout;
        } else {
            cachedClearTimeout = defaultClearTimeout;
        }
    } catch (e) {
        cachedClearTimeout = defaultClearTimeout;
    }
} ())
function runTimeout(fun) {
    if (cachedSetTimeout === setTimeout) {
        //normal enviroments in sane situations
        return setTimeout(fun, 0);
```

```
        }
        // if setTimeout wasn't available but was latter defined
        if ((cachedSetTimeout === defaultSetTimout || !cachedSetTimeout) &&
setTimeout) {
            cachedSetTimeout = setTimeout;
            return setTimeout(fun, 0);
        }
        try {
            // when when somebody has screwed with setTimeout but no I.E.
maddness
            return cachedSetTimeout(fun, 0);
        } catch(e){
            try {
                // When we are in I.E. but the script has been evaled so I.E. doesn't trust
the global object when called normally
                return cachedSetTimeout.call(null, fun, 0);
            } catch(e){
                // same as above but when it's a version of I.E. that must have the global
object for 'this', hopfully our context correct otherwise it will throw a global error
                return cachedSetTimeout.call(this, fun, 0);
            }
        }


    }
    function runClearTimeout(marker) {
```

```
        if (cachedClearTimeout === clearTimeout) {
            //normal enviroments in sane situations
            return clearTimeout(marker);
        }
        // if clearTimeout wasn't available but was latter defined
        if ((cachedClearTimeout === defaultClearTimeout || !cachedClearTimeout)
&& clearTimeout) {
            cachedClearTimeout = clearTimeout;
            return clearTimeout(marker);
        }
        try {
            // when when somebody has screwed with setTimeout but no I.E.
maddness
            return cachedClearTimeout(marker);
        } catch (e){
            try {
                // When we are in I.E. but the script has been evaled so I.E. doesn't  trust
the global object when called normally
                return cachedClearTimeout.call(null, marker);
            } catch (e){
                // same as above but when it's a version of I.E. that must have the global
object for 'this', hopfully our context correct otherwise it will throw a global error.
                // Some versions of I.E. have different rules for clearTimeout vs
setTimeout
                return cachedClearTimeout.call(this, marker);
            }
```

```
    }




}
var queue = [];
var draining = false;
var currentQueue;
var queueIndex = -1;

function cleanUpNextTick() {
    if (!draining || !currentQueue) {
        return;
    }
    draining = false;
    if (currentQueue.length) {
        queue = currentQueue.concat(queue);
    } else {
        queueIndex = -1;
    }
    if (queue.length) {
        drainQueue();
    }
}
```

```
function drainQueue() {
    if (draining) {
        return;
    }
    var timeout = runTimeout(cleanUpNextTick);
    draining = true;

    var len = queue.length;
    while(len) {
        currentQueue = queue;
        queue = [];
        while (++queueIndex < len) {
            if (currentQueue) {
                currentQueue[queueIndex].run();
            }
        }
        queueIndex = -1;
        len = queue.length;
    }
    currentQueue = null;
    draining = false;
    runClearTimeout(timeout);
}
```

```
process.nextTick = function (fun) {
    var args = new Array(arguments.length - 1);
    if (arguments.length > 1) {
        for (var i = 1; i < arguments.length; i++) {
            args[i - 1] = arguments[i];
        }
    }
    queue.push(new Item(fun, args));
    if (queue.length === 1 && !draining) {
        runTimeout(drainQueue);
    }
};

// v8 likes predictible objects
function Item(fun, array) {
    this.fun = fun;
    this.array = array;
}
Item.prototype.run = function () {
    this.fun.apply(null, this.array);
};
process.title = 'browser';
process.browser = true;
```

```
process.env = {};
process.argv = [];
process.version = ''; // empty string to avoid regexp issues
process.versions = {};

function noop() {}

process.on = noop;
process.addListener = noop;
process.once = noop;
process.off = noop;
process.removeListener = noop;
process.removeAllListeners = noop;
process.emit = noop;

process.binding = function (name) {
   throw new Error('process.binding is not supported');
};

process.cwd = function () { return '/' };
process.chdir = function (dir) {
   throw new Error('process.chdir is not supported');
};
process.umask = function() { return 0; };
```

```
},{}],14:[function(require,module,exports){
var isHexPrefixed = require('is-hex-prefixed');

/**
 * Removes '0x' from a given `String` is present
 * @param {String} str the string value
 * @return {String|Optional} a string by pass if necessary
 */
module.exports = function stripHexPrefix(str) {
  if (typeof str !== 'string') {
    return str;
  }

  return isHexPrefixed(str) ? str.slice(2) : str;
}

},{"is-hex-prefixed":10}],15:[function(require,module,exports){
// TODO: remove web3 requirement
// Call functions directly on the provider.
var Web3 = require("web3");

var Blockchain = {
  parse: function(uri) {
```

```
    var parsed = {};
    if (uri.indexOf("blockchain://") != 0) return parsed;

    uri = uri.replace("blockchain://", "");

    var pieces = uri.split("/block/");

    parsed.genesis_hash = "0x" + pieces[0];
    parsed.block_hash = "0x" + pieces[1];

    return parsed;
  },

  asURI: function(provider, callback) {
    var web3 = new Web3(provider);

    web3.eth.getBlock(0, function(err, genesis) {
      if (err) return callback(err);

      web3.eth.getBlock("latest", function(err, latest) {
        if (err) return callback(err);

        var url = "blockchain://" + genesis.hash.replace("0x", "") + "/block/" +
latest.hash.replace("0x", "");
```

```
      callback(null, url);
    });
  });
},

matches: function(uri, provider, callback) {
  uri = this.parse(uri);

  var expected_genesis = uri.genesis_hash;
  var expected_block = uri.block_hash;

  var web3 = new Web3(provider);

  web3.eth.getBlock(0, function(err, block) {
    if (err) return callback(err);
    if (block.hash != expected_genesis) return callback(null, false);

    web3.eth.getBlock(expected_block, function(err, block) {
      // Treat an error as if the block didn't exist. This is because
      // some clients respond differently.
      if (err || block == null) {
        return callback(null, false);
      }
```

```
        callback(null, true);
      });
    });
  }
};

module.exports = Blockchain;

},{"web3":5}],16:[function(require,module,exports){
var sha3 = require("crypto-js/sha3");
var schema_version = require("./package.json").version;

var TruffleSchema = {
  // Normalize options passed in to be the exact options required
  // for truffle-contract.
  //
  // options can be three things:
  // - normal object
  // - contract object
  // - solc output
  //
  // TODO: Is extra_options still necessary?
  normalizeOptions: function(options, extra_options) {
    extra_options = extra_options || {};
```

```
var normalized = {};
var expected_keys = [
  "contract_name",
  "abi",
  "binary",
  "unlinked_binary",
  "address",
  "networks",
  "links",
  "events",
  "network_id",
  "default_network",
  "updated_at"
];

// Merge options/contract object first, then extra_options
expected_keys.forEach(function(key) {
  var value;

  try {
    // Will throw an error if key == address and address doesn't exist.
    value = options[key];

    if (value != undefined) {
```

```
      normalized[key] = value;
    }
  } catch (e) {
    // Do nothing.
  }

  try {
    // Will throw an error if key == address and address doesn't exist.
    value = extra_options[key];

    if (value != undefined) {
      normalized[key] = value;
    }
  } catch (e) {
    // Do nothing.
  }
});

// Now look for solc specific items.
if (options.interface != null) {
  normalized.abi = JSON.parse(options.interface);
}

if (options.bytecode != null) {
```

```
    normalized.unlinked_binary = options.bytecode
  }

  // Assume any binary passed is the unlinked binary
  if (normalized.unlinked_binary == null && normalized.binary) {
    normalized.unlinked_binary = normalized.binary;
  }

  delete normalized.binary;

  this.copyCustomOptions(options, normalized);

  return normalized;
},

// Generate a proper binary from normalized options, and optionally
// merge it with an existing binary.
generateBinary: function(options, existing_binary, extra_options) {
  extra_options = extra_options || {};

  existing_binary = existing_binary || {};

  if (options.overwrite == true) {
    existing_binary = {};
```

```
    }

    existing_binary.contract_name = options.contract_name ||
existing_binary.contract_name || "Contract";
    existing_binary.default_network = options.default_network ||
existing_binary.default_network;

    existing_binary.abi = options.abi || existing_binary.abi;
    existing_binary.unlinked_binary = options.unlinked_binary ||
existing_binary.unlinked_binary;

    // Ensure unlinked binary starts with a 0x
    if (existing_binary.unlinked_binary &&
existing_binary.unlinked_binary.indexOf("0x") < 0) {
      existing_binary.unlinked_binary = "0x" + existing_binary.unlinked_binary;
    }

    // Merge existing networks with any passed in networks.
    existing_binary.networks = existing_binary.networks || {};
    options.networks = options.networks || {};
    Object.keys(options.networks).forEach(function(network_id) {
      existing_binary.networks[network_id] = options.networks[network_id];
    });

    var updated_at = new Date().getTime();
```

```
if (options.network_id) {
  // Ensure an object exists for this network.
  existing_binary.networks[options.network_id] =
existing_binary.networks[options.network_id] || {};

  var network = existing_binary.networks[options.network_id];

  // Override specific keys
  network.address = options.address || network.address;
  network.links = options.links;

  // merge events with any that previously existed
  network.events = network.events || {};
  options.events = options.events || {};
  Object.keys(options.events).forEach(function(event_id) {
    options.events[event_id] = options.events[event_id];
  });

  // Now overwrite any events with the most recent data from the ABI.
  existing_binary.abi.forEach(function(item) {
    if (item.type != "event") return;

    var signature = item.name + "(" + item.inputs.map(function(param) {return
param.type;}).join(",") + ")";
    network.events["0x" + sha3(signature, {outputLength: 256})] = item;
```

```
    });

    if (extra_options.dirty !== false) {
      network.updated_at = updated_at;
    }
  } else {
    if (options.address) {
      throw new Error("Cannot set address without network id");
    }
  }

  // Ensure all networks have a `links` object.
  Object.keys(existing_binary.networks).forEach(function(network_id) {
    var network = existing_binary.networks[network_id];
    network.links = network.links || {};
  });

  existing_binary.schema_version = schema_version;

  if (extra_options.dirty !== false) {
    existing_binary.updated_at = updated_at;
  } else {
    existing_binary.updated_at = options.updated_at ||
existing_binary.updated_at || updated_at;
  }
```

```
    this.copyCustomOptions(options, existing_binary);

    return existing_binary;
  },

  copyCustomOptions: function(from, to) {
    // Now let all x- options through.
    Object.keys(from).forEach(function(key) {
      if (key.indexOf("x-") != 0) return;

      try {
        value = from[key];

        if (value != undefined) {
          to[key] = value;
        }
      } catch (e) {
        // Do nothing.
      }
    });
  }
};
```

```
    module.exports = TruffleSchema;


    },{"./package.json":20,"crypto-
js/sha3":18}],17:[function(require,module,exports){
    ;(function (root, factory) {
        if (typeof exports === "object") {
            // CommonJS
            module.exports = exports = factory();
        }
        else if (typeof define === "function" && define.amd) {
            // AMD
            define([], factory);
        }
        else {
            // Global (browser)
            root.CryptoJS = factory();
        }
    }(this, function () {

        /**
         * CryptoJS core components.
         */
        var CryptoJS = CryptoJS || (function (Math, undefined) {
            /*
             * Local polyfil of Object.create
```

```
 */
var create = Object.create || (function () {
    function F() {};

    return function (obj) {
        var subtype;

        F.prototype = obj;

        subtype = new F();

        F.prototype = null;

        return subtype;
    };
}())

/**
 * CryptoJS namespace.
 */
var C = {};

/**
 * Library namespace.
```

```
 */
var C_lib = C.lib = {};

/**
 * Base object for prototypal inheritance.
 */
var Base = C_lib.Base = (function () {


    return {
        /**
         * Creates a new object that inherits from this object.
         *
         * @param {Object} overrides Properties to copy into the new object.
         *
         * @return {Object} The new object.
         *
         * @static
         *
         * @example
         *
         *     var MyType = CryptoJS.lib.Base.extend({
         *         field: 'value',
         *
```

```
 *        method: function () {
 *        }
 *    });
 */
extend: function (overrides) {
    // Spawn
    var subtype = create(this);

    // Augment
    if (overrides) {
        subtype.mixIn(overrides);
    }

    // Create default initializer
    if (!subtype.hasOwnProperty('init') || this.init === subtype.init) {
        subtype.init = function () {
            subtype.$super.init.apply(this, arguments);
        };
    }

    // Initializer's prototype is the subtype object
    subtype.init.prototype = subtype;

    // Reference supertype
```

```
    subtype.$super = this;

    return subtype;
},

/**
 * Extends this object and runs the init method.
 * Arguments to create() will be passed to init().
 *
 * @return {Object} The new object.
 *
 * @static
 *
 * @example
 *
 *     var instance = MyType.create();
 */
create: function () {
    var instance = this.extend();
    instance.init.apply(instance, arguments);

    return instance;
},
```

```
/**
 * Initializes a newly created object.
 * Override this method to add some logic when your objects are
created.
 *
 * @example
 *
 *     var MyType = CryptoJS.lib.Base.extend({
 *         init: function () {
 *             // ...
 *         }
 *     });
 */
init: function () {
},

/**
 * Copies properties into this object.
 *
 * @param {Object} properties The properties to mix in.
 *
 * @example
 *
 *     MyType.mixIn({
 *         field: 'value'
```

```
 *     });
 */
mixIn: function (properties) {
    for (var propertyName in properties) {
        if (properties.hasOwnProperty(propertyName)) {
            this[propertyName] = properties[propertyName];
        }
    }

    // IE won't copy toString using the loop above
    if (properties.hasOwnProperty('toString')) {
        this.toString = properties.toString;
    }
},

/**
 * Creates a copy of this object.
 *
 * @return {Object} The clone.
 *
 * @example
 *
 *     var clone = instance.clone();
 */
```

```
            clone: function () {
                return this.init.prototype.extend(this);
            }
        };
    }());

    /**
     * An array of 32-bit words.
     *
     * @property {Array} words The array of 32-bit words.
     * @property {number} sigBytes The number of significant bytes in this
word array.
     */
    var WordArray = C_lib.WordArray = Base.extend({
        /**
         * Initializes a newly created word array.
         *
         * @param {Array} words (Optional) An array of 32-bit words.
         * @param {number} sigBytes (Optional) The number of significant bytes
in the words.
         *
         * @example
         *
         *     var wordArray = CryptoJS.lib.WordArray.create();
```

```
 *     var wordArray = CryptoJS.lib.WordArray.create([0x00010203, 
0x04050607]);
 *     var wordArray = CryptoJS.lib.WordArray.create([0x00010203, 
0x04050607], 6);
 */
init: function (words, sigBytes) {
    words = this.words = words || [];

    if (sigBytes != undefined) {
        this.sigBytes = sigBytes;
    } else {
        this.sigBytes = words.length * 4;
    }
},

/**
 * Converts this word array to a string.
 *
 * @param {Encoder} encoder (Optional) The encoding strategy to use. 
Default: CryptoJS.enc.Hex
 *
 * @return {string} The stringified word array.
 *
 * @example
 *
```

```
 *     var string = wordArray + '';
 *     var string = wordArray.toString();
 *     var string = wordArray.toString(CryptoJS.enc.Utf8);
 */
toString: function (encoder) {
    return (encoder || Hex).stringify(this);
},

/**
 * Concatenates a word array to this word array.
 *
 * @param {WordArray} wordArray The word array to append.
 *
 * @return {WordArray} This word array.
 *
 * @example
 *
 *     wordArray1.concat(wordArray2);
 */
concat: function (wordArray) {
    // Shortcuts
    var thisWords = this.words;
    var thatWords = wordArray.words;
    var thisSigBytes = this.sigBytes;
```

```
            var thatSigBytes = wordArray.sigBytes;


            // Clamp excess bits
            this.clamp();


            // Concat
            if (thisSigBytes % 4) {
                // Copy one byte at a time
                for (var i = 0; i < thatSigBytes; i++) {
                    var thatByte = (thatWords[i >>> 2] >>> (24 - (i % 4) * 8)) & 0xff;
                    thisWords[(thisSigBytes + i) >>> 2] |= thatByte << (24 -
((thisSigBytes + i) % 4) * 8);
                }
            } else {
                // Copy one word at a time
                for (var i = 0; i < thatSigBytes; i += 4) {
                    thisWords[(thisSigBytes + i) >>> 2] = thatWords[i >>> 2];
                }
            }
            this.sigBytes += thatSigBytes;


            // Chainable
            return this;
        },
```

```
/**
 * Removes insignificant bits.
 *
 * @example
 *
 *     wordArray.clamp();
 */
clamp: function () {
    // Shortcuts
    var words = this.words;
    var sigBytes = this.sigBytes;

    // Clamp
    words[sigBytes >>> 2] &= 0xffffffff << (32 - (sigBytes % 4) * 8);
    words.length = Math.ceil(sigBytes / 4);
},

/**
 * Creates a copy of this word array.
 *
 * @return {WordArray} The clone.
 *
 * @example
 *
```

```
 *     var clone = wordArray.clone();
 */
clone: function () {
    var clone = Base.clone.call(this);
    clone.words = this.words.slice(0);

    return clone;
},

/**
 * Creates a word array filled with random bytes.
 *
 * @param {number} nBytes The number of random bytes to generate.
 *
 * @return {WordArray} The random word array.
 *
 * @static
 *
 * @example
 *
 *     var wordArray = CryptoJS.lib.WordArray.random(16);
 */
random: function (nBytes) {
    var words = [];
```

```
var r = (function (m_w) {
    var m_w = m_w;
    var m_z = 0x3ade68b1;
    var mask = 0xffffffff;

    return function () {
        m_z = (0x9069 * (m_z & 0xFFFF) + (m_z >> 0x10)) & mask;
        m_w = (0x4650 * (m_w & 0xFFFF) + (m_w >> 0x10)) & mask;
        var result = ((m_z << 0x10) + m_w) & mask;
        result /= 0x100000000;
        result += 0.5;
        return result * (Math.random() > .5 ? 1 : -1);
    }
});

for (var i = 0, rcache; i < nBytes; i += 4) {
    var _r = r((rcache || Math.random()) * 0x100000000);

    rcache = _r() * 0x3ade67b7;
    words.push((_r() * 0x100000000) | 0);
}

return new WordArray.init(words, nBytes);
```

```
    }
});

/**
 * Encoder namespace.
 */
var C_enc = C.enc = {};

/**
 * Hex encoding strategy.
 */
var Hex = C_enc.Hex = {
    /**
     * Converts a word array to a hex string.
     *
     * @param {WordArray} wordArray The word array.
     *
     * @return {string} The hex string.
     *
     * @static
     *
     * @example
     *
     *     var hexString = CryptoJS.enc.Hex.stringify(wordArray);
```

```
 */
stringify: function (wordArray) {
    // Shortcuts
    var words = wordArray.words;
    var sigBytes = wordArray.sigBytes;

    // Convert
    var hexChars = [];
    for (var i = 0; i < sigBytes; i++) {
        var bite = (words[i >>> 2] >>> (24 - (i % 4) * 8)) & 0xff;
        hexChars.push((bite >>> 4).toString(16));
        hexChars.push((bite & 0x0f).toString(16));
    }

    return hexChars.join('');
},

/**
 * Converts a hex string to a word array.
 *
 * @param {string} hexStr The hex string.
 *
 * @return {WordArray} The word array.
 *
```

```
        * @static
        *
        * @example
        *
        *     var wordArray = CryptoJS.enc.Hex.parse(hexString);
        */
       parse: function (hexStr) {
           // Shortcut
           var hexStrLength = hexStr.length;

           // Convert
           var words = [];
           for (var i = 0; i < hexStrLength; i += 2) {
               words[i >>> 3] |= parseInt(hexStr.substr(i, 2), 16) << (24 - (i % 8) * 4);
           }

           return new WordArray.init(words, hexStrLength / 2);
       }
   };

   /**
    * Latin1 encoding strategy.
    */
   var Latin1 = C_enc.Latin1 = {
```

```
/**
 * Converts a word array to a Latin1 string.
 *
 * @param {WordArray} wordArray The word array.
 *
 * @return {string} The Latin1 string.
 *
 * @static
 *
 * @example
 *
 *     var latin1String = CryptoJS.enc.Latin1.stringify(wordArray);
 */
stringify: function (wordArray) {
    // Shortcuts
    var words = wordArray.words;
    var sigBytes = wordArray.sigBytes;

    // Convert
    var latin1Chars = [];
    for (var i = 0; i < sigBytes; i++) {
        var bite = (words[i >>> 2] >>> (24 - (i % 4) * 8)) & 0xff;
        latin1Chars.push(String.fromCharCode(bite));
    }
```

```
    return latin1Chars.join('');
},

/**
 * Converts a Latin1 string to a word array.
 *
 * @param {string} latin1Str The Latin1 string.
 *
 * @return {WordArray} The word array.
 *
 * @static
 *
 * @example
 *
 *     var wordArray = CryptoJS.enc.Latin1.parse(latin1String);
 */
parse: function (latin1Str) {
    // Shortcut
    var latin1StrLength = latin1Str.length;

    // Convert
    var words = [];
    for (var i = 0; i < latin1StrLength; i++) {
```

```
            words[i >>> 2] |= (latin1Str.charCodeAt(i) & 0xff) << (24 - (i % 4) * 8);
        }

        return new WordArray.init(words, latin1StrLength);
    }
};

/**
 * UTF-8 encoding strategy.
 */
var Utf8 = C_enc.Utf8 = {
    /**
     * Converts a word array to a UTF-8 string.
     *
     * @param {WordArray} wordArray The word array.
     *
     * @return {string} The UTF-8 string.
     *
     * @static
     *
     * @example
     *
     *     var utf8String = CryptoJS.enc.Utf8.stringify(wordArray);
     */
```

```
stringify: function (wordArray) {
    try {
        return decodeURIComponent(escape(Latin1.stringify(wordArray)));
    } catch (e) {
        throw new Error('Malformed UTF-8 data');
    }
},

/**
 * Converts a UTF-8 string to a word array.
 *
 * @param {string} utf8Str The UTF-8 string.
 *
 * @return {WordArray} The word array.
 *
 * @static
 *
 * @example
 *
 *     var wordArray = CryptoJS.enc.Utf8.parse(utf8String);
 */
parse: function (utf8Str) {
    return Latin1.parse(unescape(encodeURIComponent(utf8Str)));
}
```

```
    };

    /**
     * Abstract buffered block algorithm template.
     *
     * The property blockSize must be implemented in a concrete subtype.
     *
     * @property {number} _minBufferSize The number of blocks that should
be kept unprocessed in the buffer. Default: 0
     */
    var BufferedBlockAlgorithm = C_lib.BufferedBlockAlgorithm = Base.extend({
      /**
       * Resets this block algorithm's data buffer to its initial state.
       *
       * @example
       *
       *     bufferedBlockAlgorithm.reset();
       */
      reset: function () {
        // Initial values
        this._data = new WordArray.init();
        this._nDataBytes = 0;
      },

      /**
```

```
 * Adds new data to this block algorithm's buffer.
 *
 * @param {WordArray|string} data The data to append. Strings are converted to a WordArray using UTF-8.
 *
 * @example
 *
 *     bufferedBlockAlgorithm._append('data');
 *     bufferedBlockAlgorithm._append(wordArray);
 */
_append: function (data) {
    // Convert string to WordArray, else assume WordArray already
    if (typeof data == 'string') {
        data = Utf8.parse(data);
    }

    // Append
    this._data.concat(data);
    this._nDataBytes += data.sigBytes;
},

/**
 * Processes available data blocks.
 *
```

```
* This method invokes _doProcessBlock(offset), which must be
implemented by a concrete subtype.
*
* @param {boolean} doFlush Whether all blocks and partial blocks
should be processed.
*
* @return {WordArray} The processed data.
*
* @example
*
*     var processedData = bufferedBlockAlgorithm._process();
*     var processedData = bufferedBlockAlgorithm._process(!!'flush');
*/
_process: function (doFlush) {
    // Shortcuts
    var data = this._data;
    var dataWords = data.words;
    var dataSigBytes = data.sigBytes;
    var blockSize = this.blockSize;
    var blockSizeBytes = blockSize * 4;

    // Count blocks ready
    var nBlocksReady = dataSigBytes / blockSizeBytes;
    if (doFlush) {
        // Round up to include partial blocks
```

```
            nBlocksReady = Math.ceil(nBlocksReady);
        } else {
            // Round down to include only full blocks,
            // less the number of blocks that must remain in the buffer
            nBlocksReady = Math.max((nBlocksReady | 0) - this._minBufferSize,
0);
        }

        // Count words ready
        var nWordsReady = nBlocksReady * blockSize;

        // Count bytes ready
        var nBytesReady = Math.min(nWordsReady * 4, dataSigBytes);

        // Process blocks
        if (nWordsReady) {
            for (var offset = 0; offset < nWordsReady; offset += blockSize) {
                // Perform concrete-algorithm logic
                this._doProcessBlock(dataWords, offset);
            }

            // Remove processed words
            var processedWords = dataWords.splice(0, nWordsReady);
            data.sigBytes -= nBytesReady;
        }
```

```
            // Return processed words
            return new WordArray.init(processedWords, nBytesReady);
        },

        /**
         * Creates a copy of this object.
         *
         * @return {Object} The clone.
         *
         * @example
         *
         *     var clone = bufferedBlockAlgorithm.clone();
         */
        clone: function () {
            var clone = Base.clone.call(this);
            clone._data = this._data.clone();

            return clone;
        },

        _minBufferSize: 0
    });
```

```
/**
 * Abstract hasher template.
 *
 * @property {number} blockSize The number of 32-bit words this hasher
operates on. Default: 16 (512 bits)
 */
var Hasher = C_lib.Hasher = BufferedBlockAlgorithm.extend({
   /**
    * Configuration options.
    */
   cfg: Base.extend(),

   /**
    * Initializes a newly created hasher.
    *
    * @param {Object} cfg (Optional) The configuration options to use for
this hash computation.
    *
    * @example
    *
    *     var hasher = CryptoJS.algo.SHA256.create();
    */
   init: function (cfg) {
      // Apply config defaults
      this.cfg = this.cfg.extend(cfg);
```

```
    // Set initial values
    this.reset();
},

/**
 * Resets this hasher to its initial state.
 *
 * @example
 *
 *     hasher.reset();
 */
reset: function () {
    // Reset data buffer
    BufferedBlockAlgorithm.reset.call(this);

    // Perform concrete-hasher logic
    this._doReset();
},

/**
 * Updates this hasher with a message.
 *
 * @param {WordArray|string} messageUpdate The message to append.
```

```
             *
             * @return {Hasher} This hasher.
             *
             * @example
             *
             *     hasher.update('message');
             *     hasher.update(wordArray);
             */
            update: function (messageUpdate) {
                // Append
                this._append(messageUpdate);

                // Update the hash
                this._process();

                // Chainable
                return this;
            },

            /**
             * Finalizes the hash computation.
             * Note that the finalize operation is effectively a destructive, read-once operation.
             *
```

```
             * @param {WordArray|string} messageUpdate (Optional) A final
message update.
             *
             * @return {WordArray} The hash.
             *
             * @example
             *
             *     var hash = hasher.finalize();
             *     var hash = hasher.finalize('message');
             *     var hash = hasher.finalize(wordArray);
             */
            finalize: function (messageUpdate) {
                // Final message update
                if (messageUpdate) {
                    this._append(messageUpdate);
                }

                // Perform concrete-hasher logic
                var hash = this._doFinalize();

                return hash;
            },

            blockSize: 512/32,
```

```
/**
 * Creates a shortcut function to a hasher's object interface.
 *
 * @param {Hasher} hasher The hasher to create a helper for.
 *
 * @return {Function} The shortcut function.
 *
 * @static
 *
 * @example
 *
 *     var SHA256 =
CryptoJS.lib.Hasher._createHelper(CryptoJS.algo.SHA256);
 */
_createHelper: function (hasher) {
    return function (message, cfg) {
        return new hasher.init(cfg).finalize(message);
    };
},

/**
 * Creates a shortcut function to the HMAC's object interface.
 *
 * @param {Hasher} hasher The hasher to use in this HMAC helper.
 *
```

```
             * @return {Function} The shortcut function.
             *
             * @static
             *
             * @example
             *
             *     var HmacSHA256 =
CryptoJS.lib.Hasher._createHmacHelper(CryptoJS.algo.SHA256);
             */
            _createHmacHelper: function (hasher) {
                return function (message, key) {
                    return new C_algo.HMAC.init(hasher, key).finalize(message);
                };
            }
        });

        /**
         * Algorithm namespace.
         */
        var C_algo = C.algo = {};

        return C;
    }(Math));
```

```
        return CryptoJS;


    }));
    },{}],18:[function(require,module,exports){
    ;(function (root, factory, undef) {
        if (typeof exports === "object") {
            // CommonJS
            module.exports = exports = factory(require("./core"), require("./x64-
core"));
        }
        else if (typeof define === "function" && define.amd) {
            // AMD
            define(["./core", "./x64-core"], factory);
        }
        else {
            // Global (browser)
            factory(root.CryptoJS);
        }
    }(this, function (CryptoJS) {


        (function (Math) {
            // Shortcuts
            var C = CryptoJS;
            var C_lib = C.lib;
            var WordArray = C_lib.WordArray;
```

```
var Hasher = C_lib.Hasher;
var C_x64 = C.x64;
var X64Word = C_x64.Word;
var C_algo = C.algo;

// Constants tables
var RHO_OFFSETS = [];
var PI_INDEXES  = [];
var ROUND_CONSTANTS = [];

// Compute Constants
(function () {
   // Compute rho offset constants
   var x = 1, y = 0;
   for (var t = 0; t < 24; t++) {
      RHO_OFFSETS[x + 5 * y] = ((t + 1) * (t + 2) / 2) % 64;

      var newX = y % 5;
      var newY = (2 * x + 3 * y) % 5;
      x = newX;
      y = newY;
   }

   // Compute pi index constants
```

```
for (var x = 0; x < 5; x++) {
    for (var y = 0; y < 5; y++) {
        PI_INDEXES[x + 5 * y] = y + ((2 * x + 3 * y) % 5) * 5;
    }
}

// Compute round constants
var LFSR = 0x01;
for (var i = 0; i < 24; i++) {
    var roundConstantMsw = 0;
    var roundConstantLsw = 0;

    for (var j = 0; j < 7; j++) {
        if (LFSR & 0x01) {
            var bitPosition = (1 << j) - 1;
            if (bitPosition < 32) {
                roundConstantLsw ^= 1 << bitPosition;
            } else /* if (bitPosition >= 32) */ {
                roundConstantMsw ^= 1 << (bitPosition - 32);
            }
        }

        // Compute next LFSR
        if (LFSR & 0x80) {
```

```
                    // Primitive polynomial over GF(2): x^8 + x^6 + x^5 + x^4 + 1
                    LFSR = (LFSR << 1) ^ 0x71;
                } else {
                    LFSR <<= 1;
                }
            }

            ROUND_CONSTANTS[i] = X64Word.create(roundConstantMsw,
roundConstantLsw);
        }
    }());

    // Reusable objects for temporary values
    var T = [];
    (function () {
        for (var i = 0; i < 25; i++) {
            T[i] = X64Word.create();
        }
    }());

    /**
     * SHA-3 hash algorithm.
     */
    var SHA3 = C_algo.SHA3 = Hasher.extend({
        /**
```

```
 * Configuration options.
 *
 * @property {number} outputLength
 *   The desired number of bits in the output hash.
 *   Only values permitted are: 224, 256, 384, 512.
 *   Default: 512
 */
cfg: Hasher.cfg.extend({
    outputLength: 512
}),

_doReset: function () {
    var state = this._state = []
    for (var i = 0; i < 25; i++) {
        state[i] = new X64Word.init();
    }

    this.blockSize = (1600 - 2 * this.cfg.outputLength) / 32;
},

_doProcessBlock: function (M, offset) {
    // Shortcuts
    var state = this._state;
    var nBlockSizeLanes = this.blockSize / 2;
```

```
// Absorb
for (var i = 0; i < nBlockSizeLanes; i++) {
   // Shortcuts
   var M2i  = M[offset + 2 * i];
   var M2i1 = M[offset + 2 * i + 1];

   // Swap endian
   M2i = (
      (((M2i << 8)  | (M2i >>> 24)) & 0x00ff00ff) |
      (((M2i << 24) | (M2i >>> 8))  & 0xff00ff00)
   );
   M2i1 = (
      (((M2i1 << 8)  | (M2i1 >>> 24)) & 0x00ff00ff) |
      (((M2i1 << 24) | (M2i1 >>> 8))  & 0xff00ff00)
   );

   // Absorb message into state
   var lane = state[i];
   lane.high ^= M2i1;
   lane.low  ^= M2i;
}

// Rounds
```

```
for (var round = 0; round < 24; round++) {
    // Theta
    for (var x = 0; x < 5; x++) {
        // Mix column lanes
        var tMsw = 0, tLsw = 0;
        for (var y = 0; y < 5; y++) {
            var lane = state[x + 5 * y];
            tMsw ^= lane.high;
            tLsw ^= lane.low;
        }

        // Temporary values
        var Tx = T[x];
        Tx.high = tMsw;
        Tx.low  = tLsw;
    }
    for (var x = 0; x < 5; x++) {
        // Shortcuts
        var Tx4 = T[(x + 4) % 5];
        var Tx1 = T[(x + 1) % 5];
        var Tx1Msw = Tx1.high;
        var Tx1Lsw = Tx1.low;

        // Mix surrounding columns
```

```
        var tMsw = Tx4.high ^ ((Tx1Msw << 1) | (Tx1Lsw >>> 31));
        var tLsw = Tx4.low  ^ ((Tx1Lsw << 1) | (Tx1Msw >>> 31));
        for (var y = 0; y < 5; y++) {
            var lane = state[x + 5 * y];
            lane.high ^= tMsw;
            lane.low  ^= tLsw;
        }
    }

    // Rho Pi
    for (var laneIndex = 1; laneIndex < 25; laneIndex++) {
        // Shortcuts
        var lane = state[laneIndex];
        var laneMsw = lane.high;
        var laneLsw = lane.low;
        var rhoOffset = RHO_OFFSETS[laneIndex];

        // Rotate lanes
        if (rhoOffset < 32) {
            var tMsw = (laneMsw << rhoOffset) | (laneLsw >>> (32 -
rhoOffset));
            var tLsw = (laneLsw << rhoOffset) | (laneMsw >>> (32 -
rhoOffset));
        } else /* if (rhoOffset >= 32) */ {
```

```
                var tMsw = (laneLsw << (rhoOffset - 32)) | (laneMsw >>> (64 -
rhoOffset));
                var tLsw = (laneMsw << (rhoOffset - 32)) | (laneLsw >>> (64 -
rhoOffset));
            }

            // Transpose lanes
            var TPiLane = T[PI_INDEXES[laneIndex]];
            TPiLane.high = tMsw;
            TPiLane.low  = tLsw;
        }

        // Rho pi at x = y = 0
        var T0 = T[0];
        var state0 = state[0];
        T0.high = state0.high;
        T0.low  = state0.low;

        // Chi
        for (var x = 0; x < 5; x++) {
            for (var y = 0; y < 5; y++) {
                // Shortcuts
                var laneIndex = x + 5 * y;
                var lane = state[laneIndex];
                var TLane = T[laneIndex];
```

```
            var Tx1Lane = T[((x + 1) % 5) + 5 * y];
            var Tx2Lane = T[((x + 2) % 5) + 5 * y];

            // Mix rows
            lane.high = TLane.high ^ (~Tx1Lane.high & Tx2Lane.high);
            lane.low  = TLane.low  ^ (~Tx1Lane.low  & Tx2Lane.low);
          }
        }

        // Iota
        var lane = state[0];
        var roundConstant = ROUND_CONSTANTS[round];
        lane.high ^= roundConstant.high;
        lane.low  ^= roundConstant.low;;
      }
    },

    _doFinalize: function () {
      // Shortcuts
      var data = this._data;
      var dataWords = data.words;
      var nBitsTotal = this._nDataBytes * 8;
      var nBitsLeft = data.sigBytes * 8;
      var blockSizeBits = this.blockSize * 32;
```

```
// Add padding
dataWords[nBitsLeft >>> 5] |= 0x1 << (24 - nBitsLeft % 32);
dataWords[((Math.ceil((nBitsLeft + 1) / blockSizeBits) * blockSizeBits)
>>> 5) - 1] |= 0x80;
data.sigBytes = dataWords.length * 4;

// Hash final blocks
this._process();

// Shortcuts
var state = this._state;
var outputLengthBytes = this.cfg.outputLength / 8;
var outputLengthLanes = outputLengthBytes / 8;

// Squeeze
var hashWords = [];
for (var i = 0; i < outputLengthLanes; i++) {
  // Shortcuts
  var lane = state[i];
  var laneMsw = lane.high;
  var laneLsw = lane.low;

  // Swap endian
  laneMsw = (
```

```
            (((laneMsw << 8)  | (laneMsw >>> 24)) & 0x00ff00ff) |
            (((laneMsw << 24) | (laneMsw >>> 8))  & 0xff00ff00)
        );
        laneLsw = (
            (((laneLsw << 8)  | (laneLsw >>> 24)) & 0x00ff00ff) |
            (((laneLsw << 24) | (laneLsw >>> 8))  & 0xff00ff00)
        );

        // Squeeze state to retrieve hash
        hashWords.push(laneLsw);
        hashWords.push(laneMsw);
    }

    // Return final computed hash
    return new WordArray.init(hashWords, outputLengthBytes);
},

clone: function () {
    var clone = Hasher.clone.call(this);

    var state = clone._state = this._state.slice(0);
    for (var i = 0; i < 25; i++) {
        state[i] = state[i].clone();
    }
```

```
        return clone;
    }
});

/**
 * Shortcut function to the hasher's object interface.
 *
 * @param {WordArray|string} message The message to hash.
 *
 * @return {WordArray} The hash.
 *
 * @static
 *
 * @example
 *
 *     var hash = CryptoJS.SHA3('message');
 *     var hash = CryptoJS.SHA3(wordArray);
 */
C.SHA3 = Hasher._createHelper(SHA3);

/**
 * Shortcut function to the HMAC's object interface.
 *
```

```
         * @param {WordArray|string} message The message to hash.
         * @param {WordArray|string} key The secret key.
         *
         * @return {WordArray} The HMAC.
         *
         * @static
         *
         * @example
         *
         *     var hmac = CryptoJS.HmacSHA3(message, key);
         */
        C.HmacSHA3 = Hasher._createHmacHelper(SHA3);
    }(Math));


    return CryptoJS.SHA3;

}));
},{"./core":17,"./x64-core":19}],19:[function(require,module,exports){
;(function (root, factory) {
    if (typeof exports === "object") {
        // CommonJS
        module.exports = exports = factory(require("./core"));
    }
```

```
    else if (typeof define === "function" && define.amd) {
        // AMD
        define(["./core"], factory);
    }
    else {
        // Global (browser)
        factory(root.CryptoJS);
    }
}(this, function (CryptoJS) {

    (function (undefined) {
        // Shortcuts
        var C = CryptoJS;
        var C_lib = C.lib;
        var Base = C_lib.Base;
        var X32WordArray = C_lib.WordArray;

        /**
         * x64 namespace.
         */
        var C_x64 = C.x64 = {};

        /**
         * A 64-bit word.
```

```
 */
var X64Word = C_x64.Word = Base.extend({
   /**
    * Initializes a newly created 64-bit word.
    *
    * @param {number} high The high 32 bits.
    * @param {number} low The low 32 bits.
    *
    * @example
    *
    *     var x64Word = CryptoJS.x64.Word.create(0x00010203, 0x04050607);
    */
   init: function (high, low) {
      this.high = high;
      this.low = low;
   }

   /**
    * Bitwise NOTs this word.
    *
    * @return {X64Word} A new x64-Word object after negating.
    *
    * @example
    *
```

```
 *     var negated = x64Word.not();
 */
// not: function () {
    // var high = ~this.high;
    // var low = ~this.low;

    // return X64Word.create(high, low);
// },

/**
 * Bitwise ANDs this word with the passed word.
 *
 * @param {X64Word} word The x64-Word to AND with this word.
 *
 * @return {X64Word} A new x64-Word object after ANDing.
 *
 * @example
 *
 *     var anded = x64Word.and(anotherX64Word);
 */
// and: function (word) {
    // var high = this.high & word.high;
    // var low = this.low & word.low;
```

```
    // return X64Word.create(high, low);
// },

/**
 * Bitwise ORs this word with the passed word.
 *
 * @param {X64Word} word The x64-Word to OR with this word.
 *
 * @return {X64Word} A new x64-Word object after ORing.
 *
 * @example
 *
 *     var ored = x64Word.or(anotherX64Word);
 */
// or: function (word) {
    // var high = this.high | word.high;
    // var low = this.low | word.low;

    // return X64Word.create(high, low);
// },

/**
 * Bitwise XORs this word with the passed word.
 *
```

```
 * @param {X64Word} word The x64-Word to XOR with this word.
 *
 * @return {X64Word} A new x64-Word object after XORing.
 *
 * @example
 *
 *     var xored = x64Word.xor(anotherX64Word);
 */
// xor: function (word) {
   // var high = this.high ^ word.high;
   // var low = this.low ^ word.low;

   // return X64Word.create(high, low);
// },

/**
 * Shifts this word n bits to the left.
 *
 * @param {number} n The number of bits to shift.
 *
 * @return {X64Word} A new x64-Word object after shifting.
 *
 * @example
 *
```

```
 *     var shifted = x64Word.shiftL(25);
 */
// shiftL: function (n) {
   // if (n < 32) {
      // var high = (this.high << n) | (this.low >>> (32 - n));
      // var low = this.low << n;
   // } else {
      // var high = this.low << (n - 32);
      // var low = 0;
   // }

   // return X64Word.create(high, low);
// },

/**
 * Shifts this word n bits to the right.
 *
 * @param {number} n The number of bits to shift.
 *
 * @return {X64Word} A new x64-Word object after shifting.
 *
 * @example
 *
 *     var shifted = x64Word.shiftR(7);
```

```
 */
// shiftR: function (n) {
   // if (n < 32) {
      // var low = (this.low >>> n) | (this.high << (32 - n));
      // var high = this.high >>> n;
   // } else {
      // var low = this.high >>> (n - 32);
      // var high = 0;
   // }

   // return X64Word.create(high, low);
// },

/**
 * Rotates this word n bits to the left.
 *
 * @param {number} n The number of bits to rotate.
 *
 * @return {X64Word} A new x64-Word object after rotating.
 *
 * @example
 *
 *     var rotated = x64Word.rotL(25);
 */
```

```
// rotL: function (n) {
   // return this.shiftL(n).or(this.shiftR(64 - n));
// },

/**
 * Rotates this word n bits to the right.
 *
 * @param {number} n The number of bits to rotate.
 *
 * @return {X64Word} A new x64-Word object after rotating.
 *
 * @example
 *
 *     var rotated = x64Word.rotR(7);
 */
// rotR: function (n) {
   // return this.shiftR(n).or(this.shiftL(64 - n));
// },

/**
 * Adds this word with the passed word.
 *
 * @param {X64Word} word The x64-Word to add with this word.
 *
```

```
         * @return {X64Word} A new x64-Word object after adding.
         *
         * @example
         *
         *     var added = x64Word.add(anotherX64Word);
         */
        // add: function (word) {
            // var low = (this.low + word.low) | 0;
            // var carry = (low >>> 0) < (this.low >>> 0) ? 1 : 0;
            // var high = (this.high + word.high + carry) | 0;

            // return X64Word.create(high, low);
        // }
    });

    /**
     * An array of 64-bit words.
     *
     * @property {Array} words The array of CryptoJS.x64.Word objects.
     * @property {number} sigBytes The number of significant bytes in this
word array.
     */
    var X64WordArray = C_x64.WordArray = Base.extend({
        /**
         * Initializes a newly created word array.
```

```
             *
             * @param {Array} words (Optional) An array of CryptoJS.x64.Word
objects.
             * @param {number} sigBytes (Optional) The number of significant bytes
in the words.
             *
             * @example
             *
             *     var wordArray = CryptoJS.x64.WordArray.create();
             *
             *     var wordArray = CryptoJS.x64.WordArray.create([
             *         CryptoJS.x64.Word.create(0x00010203, 0x04050607),
             *         CryptoJS.x64.Word.create(0x18191a1b, 0x1c1d1e1f)
             *     ]);
             *
             *     var wordArray = CryptoJS.x64.WordArray.create([
             *         CryptoJS.x64.Word.create(0x00010203, 0x04050607),
             *         CryptoJS.x64.Word.create(0x18191a1b, 0x1c1d1e1f)
             *     ], 10);
             */
            init: function (words, sigBytes) {
                words = this.words = words || [];

                if (sigBytes != undefined) {
                    this.sigBytes = sigBytes;
```

```
        } else {
            this.sigBytes = words.length * 8;
        }
    },

    /**
     * Converts this 64-bit word array to a 32-bit word array.
     *
     * @return {CryptoJS.lib.WordArray} This word array's data as a 32-bit
word array.
     *
     * @example
     *
     *     var x32WordArray = x64WordArray.toX32();
     */
    toX32: function () {
        // Shortcuts
        var x64Words = this.words;
        var x64WordsLength = x64Words.length;

        // Convert
        var x32Words = [];
        for (var i = 0; i < x64WordsLength; i++) {
            var x64Word = x64Words[i];
            x32Words.push(x64Word.high);
```

```
            x32Words.push(x64Word.low);
        }

        return X32WordArray.create(x32Words, this.sigBytes);
    },

    /**
     * Creates a copy of this word array.
     *
     * @return {X64WordArray} The clone.
     *
     * @example
     *
     *     var clone = x64WordArray.clone();
     */
    clone: function () {
        var clone = Base.clone.call(this);

        // Clone "words" array
        var words = clone.words = this.words.slice(0);

        // Clone each X64Word object
        var wordsLength = words.length;
        for (var i = 0; i < wordsLength; i++) {
```

```
                words[i] = words[i].clone();
            }

            return clone;
        }
    });
}());


return CryptoJS;

}));
},{"./core":17}],20:[function(require,module,exports){
module.exports={
  "_args": [
    [
      {
        "raw": "truffle-contract-schema@0.0.5",
        "scope": null,
        "escapedName": "truffle-contract-schema",
        "name": "truffle-contract-schema",
        "rawSpec": "0.0.5",
        "spec": "0.0.5",
        "type": "version"
```

},
"/Users/tim/Documents/workspace/Consensys/truffle-contract"
]
],
"_from": "truffle-contract-schema@0.0.5",
"_id": "truffle-contract-schema@0.0.5",
"_inCache": true,
"_location": "/truffle-contract-schema",
"_nodeVersion": "6.9.1",
"_npmOperationalInternal": {
"host": "packages-12-west.internal.npmjs.com",
"tmp": "tmp/truffle-contract-schema-0.0.5.tgz_1485557985137_0.46875762194395065"
},
"_npmUser": {
"name": "tcoulter",
"email": "tim@timothyjcoulter.com"
},
"_npmVersion": "3.10.8",
"_phantomChildren": {},
"_requested": {
"raw": "truffle-contract-schema@0.0.5",
"scope": null,
"escapedName": "truffle-contract-schema",
"name": "truffle-contract-schema",

"rawSpec": "0.0.5",
"spec": "0.0.5",
"type": "version"
},
"_requiredBy": [
"/"
],
"_resolved": "https://registry.npmjs.org/truffle-contract-schema/-/truffle-contract-schema-0.0.5.tgz",
"_shasum": "5e9d20bd0bf2a27fe94310748249d484eee49961",
"_shrinkwrap": null,
"_spec": "truffle-contract-schema@0.0.5",
"_where": "/Users/tim/Documents/workspace/Consensys/truffle-contract",
"author": {
"name": "Tim Coulter",
"email": "tim.coulter@consensys.net"
},
"bugs": {
"url": "https://github.com/trufflesuite/truffle-schema/issues"
},
"dependencies": {
"crypto-js": "^3.1.9-1"
},
"description": "JSON schema for contract artifacts",
"devDependencies": {

```
    "mocha": "^3.2.0"
  },
  "directories": {},
  "dist": {
    "shasum": "5e9d20bd0bf2a27fe94310748249d484eee49961",
    "tarball": "https://registry.npmjs.org/truffle-contract-schema/-/truffle-contract-schema-0.0.5.tgz"
  },
  "gitHead": "cfa4313bd4bb95bf5b94f85185203ead418f9ee6",
  "homepage": "https://github.com/trufflesuite/truffle-schema#readme",
  "keywords": [
    "ethereum",
    "json",
    "schema",
    "contract",
    "artifacts"
  ],
  "license": "MIT",
  "main": "index.js",
  "maintainers": [
    {
      "name": "tcoulter",
      "email": "tim@timothyjcoulter.com"
    }
  ],
```

"name": "truffle-contract-schema",
"optionalDependencies": {},
"readme": "ERROR: No README data found!",
"repository": {
"type": "git",
"url": "git+https://github.com/trufflesuite/truffle-schema.git"
},
"scripts": {
"test": "mocha"
},
"version": "0.0.5"
}

},{}]},{},[2]);