```
PK   ØK·X"ÈA9©      wallet-v4-code.fc#pragma version =0.2.0; ;; Wallet smart contract with plugins (slice, int) dict_get?(cell dict, int key_len, slice index) asm(index dict key_len) "DICTGET" "NULLSWAPIFNOT"; (cell, int) dict_add_builder?(cell dict, int key_len, slice index, builder value) asm(value index dict key_len) "DICTADDB"; (cell, int) dict_delete?(cell dict, int key_len, slice index) asm(index dict key_len) "DICTDEL"; () recv_internal(int my_balance, int msg_value, cell in_msg_cell, slice in_msg) impure { var cs = in_msg_cell.begin_parse(); var flags = cs~load_uint(4); ;; int_msg_info$0 ihr_disabled:Bool bounce:Bool bounced:Bool if (flags & 1) { ;; ignore all bounced messages return (); } if (in_msg.slice_bits() < 32) { ;; ignore simple transfers return (); } int op = in_msg~load_uint(32); if (op != 0x706c7567) & (op != 0x64737472) { ;; "plug" & "dstr" ;; ignore all messages not related to plugins return (); } slice s_addr = cs~load_msg_addr(); (int wc, int addr_hash) = parse_std_addr(s_addr); slice wc_n_address = begin_cell().store_int(wc, 8).store_uint(addr_hash, 256).end_cell().begin_parse(); var ds = get_data().begin_parse().skip_bits(32 + 32 + 256); var plugins = ds~load_dict(); var (_, success?) = plugins.dict_get?(8 + 256, wc_n_address); if ~(success?) { ;; it may be a transfer return (); } int query_id = in_msg~load_uint(64); var msg = begin_cell(); if (op == 0x706c7567) { ;; request funds (int r_toncoins, cell r_extra) = (in_msg~load_grams(), in_msg~load_dict()); [int my_balance, _] = get_balance(); throw_unless(80, my_balance - msg_value >= r_toncoins); msg = msg.store_uint(0x18, 6) .store_slice(s_addr) .store_grams(r_toncoins) .store_dict(r_extra) .store_uint(0, 4 + 4 + 64 + 32 + 1 + 1) .store_uint(0x706c7567 | 0x80000000, 32) .store_uint(query_id, 64); send_raw_message(msg.end_cell(), 64); } if (op == 0x64737472) { ;; remove plugin by its request plugins~dict_delete?(8 + 256, wc_n_address); var ds = get_data().begin_parse().first_bits(32 + 32 + 256); set_data(begin_cell().store_slice(ds).store_dict(plugins).end_cell()); ;; return coins only if bounce expected if (flags & 2) { msg = msg.store_uint(0x18, 6) .store_slice(s_addr) .store_grams(0) .store_uint(0, 1 + 4 + 4 + 64 + 32 + 1 + 1) .store_uint(0x64737472 | 0x80000000, 32) .store_uint(query_id, 64); send_raw_message(msg.end_cell(), 64); } } } () recv_external(slice in_msg) impure { var signature = in_msg~load_bits(512); var cs = in_msg; var (subwallet_id, valid_until, msg_seqno) = (cs~load_uint(32), cs~load_uint(32), cs~load_uint(32)); throw_if(36, valid_until <= now()); var ds = get_data().begin_parse(); var (stored_seqno, stored_subwallet, public_key, plugins) = (ds~load_uint(32), ds~load_uint(32), ds~load_uint(256), ds~load_dict()); ds.end_parse(); throw_unless(33, msg_seqno == stored_seqno); throw_unless(34, subwallet_id == stored_subwallet); throw_unless(35, check_signature(slice_hash(in_msg), signature, public_key)); accept_message(); set_data(begin_cell() .store_uint(stored_seqno + 1, 32) .store_uint(stored_subwallet, 32) .store_uint(public_key, 256) .store_dict(plugins) .end_cell()); commit(); cs~touch(); int op = cs~load_uint(8); if (op == 0) { ;; simple send while (cs.slice_refs()) { var mode = cs~load_uint(8); send_raw_message(cs~load_ref(), mode); } return (); ;; have already saved the storage } if (op == 1) { ;; deploy and install plugin int plugin_workchain = cs~load_int(8); int plugin_balance = cs~load_grams(); (cell state_init, cell body) = (cs~load_ref(), cs~load_ref()); int plugin_address = cell_hash(state_init); slice wc_n_address = begin_cell().store_int(plugin_workchain, 8).store_uint(plugin_address, 256).end_cell().begin_parse(); var msg = begin_cell() .store_uint(0x18, 6) .store_uint(4, 3).store_slice(wc_n_address) .store_grams(plugin_balance) .store_uint(4 + 2 + 1, 1 + 4 + 4 + 64 + 32 + 1 + 1 + 1) .store_ref(state_init) .store_ref(body); send_raw_message(msg.end_cell(), 3); (plugins, int success?) = plugins.dict_add_builder?(8 + 256, wc_n_address, begin_cell()); throw_unless(39, success?); } if (op == 2) { ;; install plugin slice wc_n_address = cs~load_bits(8 + 256); int amount = cs~load_grams(); int query_id = cs~load_uint(64); (plugins, int success?) = plugins.dict_add_builder?(8 + 256, wc_n_address, begin_cell()); throw_unless(39, success?); builder msg = begin_cell() .store_uint(0x18, 6) .store_uint(4, 3).store_slice(wc_n_address) .store_grams(amount) .store_uint(0, 1 + 4 + 4 + 64 + 32 + 1 + 1) .store_uint(0x6e6f7465, 32) ;; op .store_uint(query_id, 64); send_raw_message(msg.end_cell(), 3); } if (op == 3) { ;; remove plugin slice wc_n_address = cs~load_bits(8 + 256); int amount = cs~load_grams(); int query_id = cs~load_uint(64); (plugins, int success?) = plugins.dict_delete?(8 + 256, wc_n_address); throw_unless(39, success?); builder msg = begin_cell() .store_uint(0x18, 6) .store_uint(4, 3).store_slice(wc_n_address) .store_grams(amount) .store_uint(0, 1 + 4 + 4 + 64 + 32 + 1 + 1) .store_uint(0x64737472, 32) ;; op .store_uint(query_id, 64); send_raw_message(msg.end_cell(), 3); } set_data(begin_cell() .store_uint(stored_seqno + 1, 32) .store_uint(stored_subwallet, 32) .store_uint(public_key, 256) .store_dict(plugins) .end_cell()); } ;; Get methods int seqno() method_id { return get_data().begin_parse().preload_uint(32); } int get_subwallet_id() method_id { return get_data().begin_parse().skip_bits(32).preload_uint(32); } int get_public_key() method_id { var cs = get_data().begin_parse().skip_bits(64); return cs.preload_uint(256); } int is_plugin_installed(int wc, int addr_hash) method_id { var ds = get_data().begin_parse().skip_bits(32 + 32 + 256); var plugins = ds~load_dict(); var (_, success?) = plugins.dict_get?(8 + 256, begin_cell().store_int(wc, 8).store_uint(addr_hash, 256).end_cell().begin_parse()); return success?; } tuple get_plugin_list() method_id { var list = null(); var ds = get_data().begin_parse().skip_bits(32 + 32 + 256); var plugins = ds~load_dict(); do { var (wc_n_address, _, f) = plugins~dict::delete_get_min(8 + 256); if (f) { (int wc, int addr) = (wc_n_address~load_int(8), wc_n_address~load_uint(256)); list = cons(pair(wc, addr), list); } } until (~ f); return list; }PK   ØK·XoàCôk7k7 stdlib.fc;; Standard library for funC ;; forall X -> tuple cons(X head, tuple tail) asm "CONS"; forall X -> (X, tuple) uncons(tuple list) asm "UNCONS"; forall X -> (tuple, X) list_next(tuple list) asm( -> 1 0) "UNCONS"; forall X -> X car(tuple list) asm "CAR"; tuple cdr(tuple list) asm "CDR"; tuple empty_tuple() asm "NIL"; forall X -> tuple tpush(tuple t, X value) asm "TPUSH"; forall X -> (tuple, ()) ~tpush(tuple t, X value) asm "TPUSH"; forall X -> [X] single(X x) asm "SINGLE"; forall X -> X unsingle([X] t) asm "UNSINGLE"; forall X, Y -> [X, Y] pair(X x, Y y) asm "PAIR"; forall X, Y -> (X, Y) unpair([X, Y] t) asm "UNPAIR"; forall X, Y, Z -> [X, Y, Z] triple(X x, Y y, Z z) asm "TRIPLE"; forall X, Y, Z -> (X, Y, Z) untriple([X, Y, Z] t) asm "UNTRIPLE"; forall X, Y, Z, W -> [X, Y, Z, W] tuple4(X x, Y y, Z z, W w) asm "4 TUPLE"; forall X, Y, Z, W -> (X, Y, Z, W) untuple4([X, Y, Z, W] t) asm "4 UNTUPLE"; forall X -> X first(tuple t) asm "FIRST"; forall X -> X second(tuple t) asm "SECOND"; forall X -> X third(tuple t) asm "THIRD"; forall X -> X fourth(tuple t) asm "3 INDEX"; forall X, Y -> X pair_first([X, Y] p) asm "FIRST"; forall X, Y -> Y pair_second([X, Y] p) asm "SECOND"; forall X, Y, Z -> X triple_first([X, Y, Z] p) asm "FIRST"; forall X, Y, Z -> Y triple_second([X, Y, Z] p) asm "SECOND"; forall X, Y, Z -> Z triple_third([X, Y, Z] p) asm "THIRD"; forall X -> X null() asm "PUSHNULL"; forall X -> (X, ()) ~impure_touch(X x) impure asm "NOP"; int now() asm "NOW"; slice my_address() asm "MYADDR"; [int, cell] get_balance() asm "BALANCE"; int cur_lt() asm "LTIME"; int block_lt() asm "BLOCKLT"; int cell_hash(cell c) asm "HASHCU"; int slice_hash(slice s) asm "HASHSU"; int string_hash(slice s) asm "SHA256U"; int check_signature(int hash, slice signature, int public_key) asm "CHKSIGNU"; int check_data_signature(slice data, slice signature, int public_key) asm "CHKSIGNS"; (int, int, int) compute_data_size(cell c, int max_cells) impure asm "CDATASIZE"; (int, int, int) slice_compute_data_size(slice s, int max_cells) impure asm "SDATASIZE"; (int, int, int, int) compute_data_size?(cell c, int max_cells) asm "CDATASIZEQ NULLSWAPIFNOT2 NULLSWAPIFNOT"; (int, int, int, int) slice_compute_data_size?(cell c, int max_cells) asm "SDATASIZEQ NULLSWAPIFNOT2 NULLSWAPIFNOT"; ;; () throw_if(int excno, int cond) impure asm "THROWARGIF"; () dump_stack() impure asm "DUMPSTK"; cell get_data() asm "c4 PUSH"; () set_data(cell c) impure asm "c4 POP"; cont get_c3() impure asm "c3 PUSH"; () set_c3(cont c) impure asm "c3 POP"; cont bless(slice s) impure asm "BLESS"; () accept_message() impure asm "ACCEPT"; () set_gas_limit(int limit) impure asm "SETGASLIMIT"; () commit() impure asm "COMMIT"; () buy_gas(int gram) impure asm "BUYGAS"; int min(int x, int y) asm "MIN"; int max(int x, int y) asm "MAX"; (int, int) minmax(int x, int y) asm "MINMAX"; int abs(int x) asm "ABS"; slice begin_parse(cell c) asm "CTOS"; () end_parse(slice s) impure asm "ENDS"; (slice, cell) load_ref(slice s) asm( -> 1 0) "LDREF"; cell preload_ref(slice s) asm "PLDREF"; ;; (slice, int) ~load_int(slice s, int len) asm(s len -> 1 0) "LDIX"; ;; (slice, int) ~load_uint(slice s, int len) asm( -> 1 0) "LDUX"; ;; int preload_int(slice s, int len) asm "PLDIX"; ;; int preload_uint(slice s, int len) asm "PLDUX"; ;; (slice, slice) load_bits(slice s, int len) asm(s len -> 1 0) "LDSLICEX"; ;; slice preload_bits(slice s, int len) asm "PLDSLICEX"; (slice, int) load_grams(slice s) asm( -> 1 0) "LDGRAMS"; slice skip_bits(slice s, int len) asm "SDSKIPFIRST"; (slice, ()) ~skip_bits(slice s, int len) asm "SDSKIPFIRST"; slice first_bits(slice s, int len) asm "SDCUTFIRST"; slice skip_last_bits(slice s, int len) asm "SDSKIPLAST"; (slice, ()) ~skip_last_bits(slice s, int len) asm "SDSKIPLAST"; slice slice_last(slice s, int len) asm "SDCUTLAST"; (slice, cell) load_dict(slice s) asm( -> 1 0) "LDDICT"; cell preload_dict(slice s) asm "PLDDICT"; slice skip_dict(slice s) asm "SKIPDICT"; (slice, cell) load_maybe_ref(slice s) asm( -> 1 0) "LDOPTREF"; cell preload_maybe_ref(slice s) asm "PLDOPTREF"; builder store_maybe_ref(builder b, cell c) asm(c b) "STOPTREF"; int cell_depth(cell c) asm "CDEPTH"; int slice_refs(slice s) asm "SREFS"; int slice_bits(slice s) asm "SBITS"; (int, int) slice_bits_refs(slice s) asm "SBITREFS"; int slice_empty?(slice s) asm "SEMPTY"; int slice_data_empty?(slice s) asm "SDEMPTY"; int slice_refs_empty?(slice s) asm "SREMPTY"; int slice_depth(slice s) asm "SDEPTH"; int builder_refs(builder b) asm "BREFS"; int builder_bits(builder b) asm "BBITS"; int builder_depth(builder b) asm "BDEPTH"; builder begin_cell() asm "NEWC"; cell end_cell(builder b) asm "ENDC"; builder store_ref(builder b, cell c) asm(c b) "STREF"; ;; builder store_uint(builder b, int x, int len) asm(x b len) "STUX"; ;; builder store_int(builder b, int x, int len) asm(x b len) "STIX"; builder store_slice(builder b, slice s) asm "STSLICER"; builder store_grams(builder b, int x) asm "STGRAMS"; builder store_dict(builder b, cell c) asm(c b) "STDICT"; (slice, slice) load_msg_addr(slice s) asm( -> 1 0) "LDMSGADDR"; tuple parse_addr(slice s) asm "PARSEMSGADDR"; (int, int) parse_std_addr(slice s) asm "REWRITESTDADDR"; (int, slice) parse_var_addr(slice s) asm "REWRITEVARADDR"; cell idict_set_ref(cell dict, int key_len, int index, cell value) asm(value index dict key_len) "DICTISETREF"; (cell, ()) ~idict_set_ref(cell dict, int key_len, int index, cell value) asm(value index dict key_len) "DICTISETREF"; cell udict_set_ref(cell dict, int key_len, int index, cell value) asm(value index dict key_len) "DICTUSETREF"; (cell, ()) ~udict_set_ref(cell dict, int key_len, int index, cell value) asm(value index dict key_len) "DICTUSETREF"; cell idict_get_ref(cell dict, int key_len, int index) asm(index dict key_len) "DICTIGETOPTREF"; (cell, int) idict_get_ref?(cell dict, int key_len, int index) asm(index dict key_len) "DICTIGETREF"; (cell, int) udict_get_ref?(cell dict, int key_len, int index) asm(index dict key_len) "DICTUGETREF"; (cell, cell) idict_set_get_ref(cell dict, int key_len, int index, cell value) asm(value index dict key_len) "DICTISETGETOPTREF"; (cell, cell) udict_set_get_ref(cell dict, int key_len, int index, cell value) asm(value index dict key_len) "DICTUSETGETOPTREF"; (cell, int) idict_delete?(cell dict, int key_len, int index) asm(index dict key_len) "DICTIDEL"; (cell, int) udict_delete?(cell dict, int key_len, int index) asm(index dict key_len) "DICTUDEL"; (slice, int) idict_get?(cell dict, int key_len, int index) asm(index dict key_len) "DICTIGET" "NULLSWAPIFNOT"; (slice, int) udict_get?(cell dict, int key_len, int index) asm(index dict key_len) "DICTUGET" "NULLSWAPIFNOT"; (cell, slice, int) idict_delete_get?(cell dict, int key_len, int index) asm(index dict key_len) "DICTIDELGET" "NULLSWAPIFNOT"; (cell, slice, int) udict_delete_get?(cell dict, int key_len, int index) asm(index dict key_len) "DICTUDELGET" "NULLSWAPIFNOT"; (cell, (slice, int)) ~idict_delete_get?(cell dict, int key_len, int index) asm(index dict key_len) "DICTIDELGET" "NULLSWAPIFNOT"; (cell, (slice, int)) ~udict_delete_get?(cell dict, int key_len, int index) asm(index dict key_len) "DICTUDELGET" "NULLSWAPIFNOT"; cell udict_set(cell dict, int key_len, int index, slice value) asm(value index dict key_len) "DICTUSET"; (cell, ()) ~udict_set(cell dict, int key_len, int index, slice value) asm(value index dict key_len) "DICTUSET"; cell idict_set(cell dict, int key_len, int index, slice value) asm(value index dict key_len) "DICTISET"; (cell, ()) ~idict_set(cell dict, int key_len, int index, slice value) asm(value index dict key_len) "DICTISET"; cell dict_set(cell dict, int key_len, slice index, slice value) asm(value index dict key_len) "DICTSET"; (cell, ()) ~dict_set(cell dict, int key_len, slice index, slice value) asm(value index dict key_len) "DICTSET"; (cell, int) udict_add?(cell dict, int key_len, int index, slice value) asm(value index dict key_len) "DICTUADD"; (cell, int) udict_replace?(cell dict, int key_len, int index, slice value) asm(value index dict key_len) "DICTUREPLACE"; (cell, int) idict_add?(cell dict, int key_len, int index, slice value) asm(value index dict key_len) "DICTIADD"; (cell, int) idict_replace?(cell dict, int key_len, int index, slice value) asm(value index dict key_len) "DICTIREPLACE"; cell udict_set_builder(cell dict, int key_len, int index, builder value) asm(value index dict key_len) "DICTUSETB"; (cell, ()) ~udict_set_builder(cell dict, int key_len, int index, builder value) asm(value index dict key_len) "DICTUSETB"; cell idict_set_builder(cell dict, int key_len, int index, builder value) asm(value index dict key_len) "DICTISETB"; (cell, ()) ~idict_set_builder(cell dict, int key_len, int index, builder value) asm(value index dict key_len) "DICTISETB"; cell dict_set_builder(cell dict, int key_len, slice index, builder value) asm(value index dict key_len) "DICTSETB"; (cell, ()) ~dict_set_builder(cell dict, int key_len, slice index, builder value) asm(value index dict key_len) "DICTSETB"; (cell, int) udict_add_builder?(cell dict, int key_len, int index, builder value) asm(value index dict key_len) "DICTUADDB"; (cell, int) udict_replace_builder?(cell dict, int key_len, int index, builder value) asm(value index dict key_len) "DICTUREPLACEB"; (cell, int) idict_add_builder?(cell dict, int key_len, int index, builder value) asm(value index dict key_len) "DICTIADDB"; (cell, int) idict_replace_builder?(cell dict, int key_len, int index, builder value) asm(value index dict key_len) "DICTIREPLACEB"; (cell, int, slice, int) udict_delete_get_min(cell dict, int key_len) asm(-> 0 2 1 3) "DICTUREMMIN" "NULLSWAPIFNOT2"; (cell, (int, slice, int)) ~udict::delete_get_min(cell dict, int key_len) asm(-> 0 2 1 3) "DICTUREMMIN" "NULLSWAPIFNOT2"; (cell, int, slice, int) idict_delete_get_min(cell dict, int key_len) asm(-> 0 2 1 3) "DICTIREMMIN" "NULLSWAPIFNOT2"; (cell, (int, slice, int)) ~idict::delete_get_min(cell dict, int key_len) asm(-> 0 2 1 3) "DICTIREMMIN" "NULLSWAPIFNOT2"; (cell, slice, slice, int) dict_delete_get_min(cell dict, int key_len) asm(-> 0 2 1 3) "DICTREMMIN" "NULLSWAPIFNOT2"; (cell, (slice, slice, int)) ~dict::delete_get_min(cell dict, int key_len) asm(-> 0 2 1 3) "DICTREMMIN" "NULLSWAPIFNOT2"; (cell, int, slice, int) udict_delete_get_max(cell dict, int key_len) asm(-> 0 2 1 3) "DICTUREMMAX" "NULLSWAPIFNOT2"; (cell, (int, slice, int)) ~udict::delete_get_max(cell dict, int key_len) asm(-> 0 2 1 3) "DICTUREMMAX" "NULLSWAPIFNOT2"; (cell, int, slice, int) idict_delete_get_max(cell dict, int key_len) asm(-> 0 2 1 3) "DICTIREMMAX" "NULLSWAPIFNOT2"; (cell, (int, slice, int)) ~idict::delete_get_max(cell dict, int key_len) asm(-> 0 2 1 3) "DICTIREMMAX" "NULLSWAPIFNOT2"; (cell, slice, slice, int) dict_delete_get_max(cell dict, int key_len) asm(-> 0 2 1 3) "DICTREMMAX" "NULLSWAPIFNOT2"; (cell, (slice, slice, int)) ~dict::delete_get_max(cell dict, int key_len) asm(-> 0 2 1 3) "DICTREMMAX" "NULLSWAPIFNOT2"; (int, slice, int) udict_get_min?(cell dict, int key_len) asm (-> 1 0 2) "DICTUMIN" "NULLSWAPIFNOT2"; (int, slice, int) udict_get_max?(cell dict, int key_len) asm (-> 1 0 2) "DICTUMAX" "NULLSWAPIFNOT2"; (int, cell, int) udict_get_min_ref?(cell dict, int key_len) asm (-> 1 0 2) "DICTUMINREF" "NULLSWAPIFNOT2"; (int, cell, int) udict_get_max_ref?(cell dict, int key_len) asm (-> 1 0 2) "DICTUMAXREF" "NULLSWAPIFNOT2"; (int, slice, int) idict_get_min?(cell dict, int key_len) asm (-> 1 0 2) "DICTIMIN" "NULLSWAPIFNOT2"; (int, slice, int) idict_get_max?(cell dict, int key_len) asm (-> 1 0 2) "DICTIMAX" "NULLSWAPIFNOT2"; (int, cell, int) idict_get_min_ref?(cell dict, int key_len) asm (-> 1 0 2) "DICTIMINREF" "NULLSWAPIFNOT2"; (int, cell, int) idict_get_max_ref?(cell dict, int key_len) asm (-> 1 0 2) "DICTIMAXREF" "NULLSWAPIFNOT2"; (int, slice, int) udict_get_next?(cell dict, int key_len, int pivot) asm(pivot dict key_len -> 1 0 2) "DICTUGETNEXT" "NULLSWAPIFNOT2"; (int, slice, int) udict_get_nexteq?(cell dict, int key_len, int pivot) asm(pivot dict key_len -> 1 0 2) "DICTUGETNEXTEQ" "NULLSWAPIFNOT2"; (int, slice, int) udict_get_prev?(cell dict, int key_len, int pivot) asm(pivot dict key_len -> 1 0 2) "DICTUGETPREV" "NULLSWAPIFNOT2"; (int, slice, int) udict_get_preveq?(cell dict, int key_len, int pivot) asm(pivot dict key_len -> 1 0 2) "DICTUGETPREVEQ" "NULLSWAPIFNOT2"; (int, slice, int) idict_get_next?(cell dict, int key_len, int pivot) asm(pivot dict key_len -> 1 0 2) "DICTIGETNEXT" "NULLSWAPIFNOT2"; (int, slice, int) idict_get_nexteq?(cell dict, int key_len, int pivot) asm(pivot dict key_len -> 1 0 2) "DICTIGETNEXTEQ" "NULLSWAPIFNOT2"; (int, slice, int) idict_get_prev?(cell dict, int key_len, int pivot) asm(pivot dict key_len -> 1 0 2) "DICTIGETPREV" "NULLSWAPIFNOT2"; (int, slice, int) idict_get_preveq?(cell dict, int key_len, int pivot) asm(pivot dict key_len -> 1 0 2) "DICTIGETPREVEQ" "NULLSWAPIFNOT2"; cell new_dict() asm "NEWDICT"; int dict_empty?(cell c) asm "DICTEMPTY"; (slice, slice, slice, int) pfxdict_get?(cell dict, int key_len, slice key) asm(key dict key_len) "PFXDICTGETQ" "NULLSWAPIFNOT2"; (cell, int) pfxdict_set?(cell dict, int key_len, slice key, slice value) asm(value key dict key_len) "PFXDICTSET"; (cell, int) pfxdict_delete?(cell dict, int key_len, slice key) asm(key dict key_len) "PFXDICTDEL"; cell config_param(int x) asm "CONFIGOPTPARAM"; int cell_null?(cell c) asm "ISNULL"; () raw_reserve(int amount, int mode) impure asm "RAWRESERVE"; () raw_reserve_extra(int amount, cell extra_amount, int mode) impure asm "RAWRESERVEX"; () send_raw_message(cell msg, int mode) impure asm "SENDRAWMSG"; () set_code(cell new_code) impure asm "SETCODE"; int random() impure asm "RANDU256"; int rand(int range) impure asm "RAND"; int get_seed() impure asm "RANDSEED"; int set_seed() impure asm "SETRAND"; () randomize(int x) impure asm "ADDRAND"; () randomize_lt() impure asm "LTIME" "ADDRAND"; builder store_coins(builder b, int x) asm "STVARUINT16"; (slice, int) load_coins(slice s) asm( -> 1 0) "LDVARUINT16"; int equal_slices (slice a, slice b) asm "SDEQ"; int builder_null?(builder b) asm "ISNULL"; builder store_builder(builder to, builder from) asm "STBR"; PK     ØK·X"ÈA9©      wallet-v4-code.fcPK     ØK·XoàCôk7k7 Ø stdlib.fcPK      pjR
```