

# Report: Decentralized Inheritance Protocol

Noah Klaholz, Vincent Schall, Max Mendes Carvalho

November 2025

## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                              | <b>2</b>  |
| 1.1      | Motivation . . . . .                             | 2         |
| 1.2      | The Decentralized Inheritance Protocol . . . . . | 2         |
| <b>2</b> | <b>Smart Contract architecture</b>               | <b>2</b>  |
| <b>3</b> | <b>Appendix</b>                                  | <b>2</b>  |
|          | Appendices                                       | 2         |
| <b>A</b> | <b>References</b>                                | <b>11</b> |

# 1 Introduction

## 1.1 Motivation

No one can escape death - but what happens to your crypto when you die? According to [1], it is estimated that around 3.7 million Bitcoin are lost and unrecoverable. One of the top reasons is death: crypto holders that passed away and failed to share access information with heirs will be responsible for inaccessible funds.

Traditional inheritance systems are flawed: they take very long, are expensive and more often than not lead to conflict between the heirs. We want to solve these problems by introducing a decentralized inheritance protocol.

## 1.2 The Decentralized Inheritance Protocol

The idea is as follows: anyone can create a will by deploying the inheritance protocol contract. After that, depositing coins, tokens and assets, as well as defining beneficiaries or heirs by adding their wallet addresses, is quick and easy with function calls to the contract. For each beneficiary, the owner can define a payout amount as a percentage of the total deposited assets.

Furthermore, deposited assets are invested using Aave<sup>1</sup>. This allows the balance to grow instead of laying dry.

The owner has to check in at least every 90 days to verify that he's still alive. As long as these check-ins occur, there will be no payout. When a check in is missed there is a 30-day grace period during which a check in can be made again **TODO** In case of death, trusted oracles (in most cases a notary) are used to verify the death via death certificates, before initiating the payout.

## 2 Smart Contract architecture

## 3 Appendix

# Appendices

```
1      // SPDX-License-Identifier: MIT
2      pragma solidity ^0.8.28;
3
4
5      import "@openzeppelin/contracts/access/Ownable.sol";
6      import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
7      import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
8      import {IDeathOracle} from "./mocks/IDeathOracle.sol";
9      import {MockAavePool} from "./mocks/MockAavePool.sol";
10
11     contract InheritanceProtocol is Ownable, ReentrancyGuard {
12
13         IERC20 public immutable usdc;
14         IDeathOracle public immutable deathOracle;
15         address private notaryAddress;
16         MockAavePool public aavePool;
```

---

<sup>1</sup>Aave — a decentralized lending protocol: supply crypto to earn interest via liquidity pools. <https://aave.com/docs/developers/liquidity-pool>

```

17
18     // address for donations (underdetermined payout)
19     address private ourAddress;
20
21     /**
22      * Stores address and payout percentage amount (0-100) of
23      * a beneficiary.
24     */
25     struct Beneficiary {
26         address payoutAddress;
27         uint256 amount;
28     }
29
30     Beneficiary[10] private _beneficiaries;
31
32     State private _currentState;
33
34     uint256 private _lastCheckIn;
35     bool private _called = false;
36
37     uint256 private constant NOT_FOUND = type(uint256).max;
38     uint256 private constant MAX_BENEFICIARIES = 10;
39     uint256 private constant MAX_PERCENTAGE = 100;
40     uint256 private constant CHECK_IN_PERIOD = 90 * 1 days;
41     uint256 private constant GRACE_PERIOD = 30 * 1 days;
42
43     event BeneficiaryAdded(address indexed payoutAddress,
44                           uint256 amount, uint256 index);
45     event BeneficiaryRemoved(address indexed payoutAddress,
46                           uint256 index);
47     event Deposited(uint256 amount);
48     event Withdrawn(uint256 amount);
49     event CheckedIn(uint256 timestamp);
50     event StateChanged(uint256 timestamp, State from, State
51                      to);
52     event PayoutMade(uint256 amount, address payoutAddress);
53     event TestEvent(string s);
54     event TestEventNum(uint s);
55
56     /**
57      * Initializes a new InheritanceProtocol.
58      * @param _usdcAddress address of the currency used
59      * (non-zero).
60     */
61     constructor(address _usdcAddress, address
62                 _deathOracleAddress, address _notaryAddress, address
63                 _aavePoolAddress) Ownable(msg.sender) {
64         require(_usdcAddress != address(0), "USDC address
zero");
65         require(_deathOracleAddress != address(0), "Death
Oracle address zero");
66         ourAddress =
67             0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266;
68         usdc = IERC20(_usdcAddress);
69         deathOracle = IDeathOracle(_deathOracleAddress);
70         notaryAddress = _notaryAddress;
71         aavePool = MockAavePool(_aavePoolAddress);
72         _currentState = State.ACTIVE;

```

```

65         _lastCheckIn = block.timestamp;
66     }
67
68     ////////////////////////////////////////////////////////////////// MODIFIERS ///////////////////
69
70     /**
71      * This modifier requires the function call to be made
72      * before distribution.
73     */
74     modifier onlyPreDistribution() {
75         require(_currentState < State.DISTRIBUTION, "Cannot
76             modify funds post-distribution");
77         -;
78     }
79
80     /**
81      * This modifier requires the function call to be made in
82      * the ACTIVE or WARNING phase
83     */
84     modifier onlyActiveWarning() {
85         require(_currentState < State.VERIFICATION, "Cannot
86             make administrative changes without Owner
87             check-In");
88         -;
89     }
90
91     /**
92      * This modifier requires the function call to be made in
93      * the DISTRIBUTION phase
94     */
95     modifier onlyDistribution() {
96         require(_currentState == State.DISTRIBUTION, "Can only
97             make payouts in distribution phase");
98         -;
99     }
100
101    /**
102      * This modifier requires the function call to be made by
103      * the notary
104     */
105    modifier onlyNotary() {
106        require(msg.sender == notaryAddress, "Only notary can
107            call this function");
108        -;
109    }
110
111    ////////////////////////////////////////////////////////////////// STATE MACHINE & CHECK-INS ///////////////////
112
113    /**
114      * Defines the state of the contract.
115      * - Active: mutable state, owner check-ins required.
116      * - Warning: Missed check-in, notification sent at 90
117      * days,
118      * verification phase starts at 120 days.
119      * - Verification: submission of death certificate (30
120      * days).
121      * - Distribution: distribute assets based on defined
122      * conditions.

```

```

111     */
112     enum State { ACTIVE, WARNING, VERIFICATION, DISTRIBUTION }
113
114     /**
115      * Updates the State in the State-Machine
116      * Should always be possible and accessible by anyone
117      * @return currentState after execution
118      */
119     function updateState() public returns (State) {
120         uint256 elapsed = uint256(block.timestamp) -
121             _lastCheckIn;
122         State oldState = _currentState;
123
124         // --- Phase transitions in logical order ---
125
126         // If in ACTIVE and check-in expired           WARNING
127         if (_currentState == State.ACTIVE && elapsed >
128             CHECK_IN_PERIOD) {
129             _currentState = State.WARNING;
130         }
131
132         // If in WARNING and grace period expired     VERIFICATION
133         if (_currentState == State.WARNING && elapsed >
134             CHECK_IN_PERIOD + GRACE_PERIOD) {
135             _currentState = State.VERIFICATION;
136         }
137
138         // If in VERIFICATION and death confirmed    DISTRIBUTION
139         if (_currentState == State.VERIFICATION &&
140             deathOracle.isDeceased(owner())) {
141             _currentState = State.DISTRIBUTION;
142         }
143
144         emit StateChanged(block.timestamp, oldState,
145                           _currentState);
146
147         // Trigger payout if we reached DISTRIBUTION
148         if (_currentState == State.DISTRIBUTION) {
149             distributePayout();
150         }
151
152         return _currentState;
153     }
154
155     /**
156      * Changes the state of the contract to a given state.
157      * @param to the state to change to.
158      */
159     function changeState (State to) public {
160         require(to != _currentState, "Already in requested
161                 state");
162         emit StateChanged(block.timestamp, _currentState, to);
163         _currentState = to;
164     }
165
166     /**

```

```

161     * The owner checks in to verify that he's alive.
162     * Should be possible in active and warning state.
163     */
164     function checkIn() public onlyOwner {
165         require(_currentState == State.ACTIVE || _currentState
166             == State.WARNING, "Need to be in active or warning
167             state");
168         emit CheckedIn(block.timestamp);
169         _lastCheckIn = block.timestamp;
170     }
171
172     /**
173      * Finds the index of a beneficiary in the beneficiaries
174      * list.
175      * @param _address the address whose index to find.
176      * @return the index if the address is in the list,
177      * 'NOT_FOUND' otherwise.
178      */
179     function findBeneficiaryIndex(address _address) public
180         view returns (uint256) {
181         if (_address == address(0)) {
182             return NOT_FOUND;
183         }
184         for (uint256 i = 0; i < MAX_BENEFICIARIES; i++) {
185             if (_beneficiaries[i].payoutAddress == _address) {
186                 return i;
187             }
188         }
189         return NOT_FOUND;
190     }
191     /**
192      * Removes a beneficiary with a given address.
193      * Only the owner can perform this action.
194      * @param _address the address to remove.
195      * Fails if the provided address is zero OR not in the
196      * list of beneficiaries.
197      * @return true if the deletion was successful, false
198      * otherwise.
199      */
200     function removeBeneficiary(address _address) public
201         onlyOwner onlyActiveWarning returns (bool) {
202         checkIn();
203         uint256 index = findBeneficiaryIndex(_address);
204         if (index == NOT_FOUND) {
205             return false;
206         }
207         delete _beneficiaries[index];
208         emit BeneficiaryRemoved(_address, index);
209         return true;
210     }
211
212     /**
213      * Adds a beneficiary to the list.
214      * Only the owner can perform this action.
215      * Requirements:

```

```

211     * - List not full
212     * - Payout after adding <= 100
213     * @param _address the address to add to the list.
214     * @param _amount the payout amount related to this
215         address.
216     * @return true if the addition was successful, false
217         otherwise.
218     */
219     function addBeneficiary(address _address, uint256 _amount)
220         public onlyOwner onlyActiveWarning returns (bool) {
221         checkIn();
222         require(_address != address(0), "Invalid address");
223         require(_amount > 0 && _amount <= MAX_PERCENTAGE,
224             "Invalid amount");
225
226
227         // Check for duplicate
228         if (findBeneficiaryIndex(_address) != NOT_FOUND) {
229             return false;
230         }
231
232
233         // Find empty slot
234         uint256 emptyIndex = NOT_FOUND;
235         for (uint256 i = 0; i < MAX_BENEFICIARIES; i++) {
236             if (_beneficiaries[i].payoutAddress == address(0))
237             {
238                 emptyIndex = i;
239                 break;
240             }
241
242             if (emptyIndex == NOT_FOUND) {
243                 return false; // Max beneficiaries reached
244             }
245
246             _beneficiaries[emptyIndex] = Beneficiary({
247                 payoutAddress: _address, amount: _amount });
248             emit BeneficiaryAdded(_address, _amount, emptyIndex);
249             return true;
250         }
251
252         /**
253         * Deposits a given amount of USDC.
254         * @param _amount the amount to deposit.
255         */
256         function deposit(uint256 _amount) external onlyOwner
257             nonReentrant onlyPreDistribution {
258             checkIn();
259             require(_amount > 0, "Amount has to be greater than
zero.");

```

```

260         usdc.transferFrom(msg.sender, address(this), _amount);
261
262         usdc.approve(address(aavePool), _amount);
263
264         aavePool.supply(address(usdc), _amount, address(this));
265
266         emit Deposited(_amount);
267     }
268
269
270     /**
271      * Withdraws a given amount of USDC.
272      * @param _amount the amount to withdraw.
273      */
274     function withdraw(uint256 _amount) external onlyOwner
275         nonReentrant onlyPreDistribution {
276         checkIn();
277         require(_amount > 0, "Amount has to be greater than
278             zero.");
279         require(getBalance() >= _amount, "Insufficient
280             balance");
281
282         aavePool.withdraw(address(usdc), _amount,
283             address(this));
284
285         usdc.transfer(msg.sender, _amount);
286         emit Withdrawn(_amount);
287     }
288
289     ///////////////////////////////////////////////////////////////////
290
291     /**
292      * Upload the death verification to the chain
293      * Only callable by the notary
294      */
295     function uploadDeathVerification(bool _deceased, bytes
296         calldata _proof) external onlyNotary{
297         deathOracle.setDeathStatus(owner(), _deceased, _proof);
298     }
299
300     /**
301      * Checks if the owner died by calling death certificate
302      * oracle.
303      * @return true if the owner died, else otherwise.
304      */
305     function checkIfOwnerDied() public view returns (bool) {
306         return deathOracle.isDeceased(owner());
307     }
308
309     ///////////////////////////////////////////////////////////////////
310
311     /**
312      * Distributes the payout based on definitions given by
313      * owner.
314      * Is only called in the updateState() Function, after
315      * death verification
316      */
317     function distributePayout() public {

```

```

310     require(!_called, "Payout can only be called once.");
311     _called = true;
312     bool donation = !isPayoutFullyDetermined();
313     uint256 count = getActiveCount();
314     Beneficiary[] memory activeBeneficiaries =
315         getActiveBeneficiaries();
316     uint256 balanceRemainingInPool = getBalance();
317     uint256 originalBalance =
318         aavePool.withdraw(address(usdc),
319             balanceRemainingInPool, address(this));
320     for (uint256 i=0; i<count; i++) {
321         Beneficiary memory beneficiary =
322             activeBeneficiaries[i];
323         uint256 amount = beneficiary.amount;
324         address payoutAddress = beneficiary.payoutAddress;
325
326         uint actualAmount = (originalBalance * amount) /
327             MAX_PERCENTAGE;
328
329         usdc.transfer( payoutAddress, actualAmount);
330         emit PayoutMade(actualAmount, payoutAddress);
331     }
332     if (donation) {
333         // If the payout is not fully determined, the rest
334         // of the balance will be sent to the developer
335         // team.
336         // For now this is hardcoded as the first address
337         // generated by hardhat when running a local node.
338         uint256 donatedAmount =
339             aavePool.withdraw(address(usdc), getBalance(),
340                 address(this));
341         usdc.transfer(ourAddress, donatedAmount);
342         emit PayoutMade(donatedAmount, ourAddress);
343     }
344
345     /**
346      * Checks if the currently defined payout is fully
347      * determined, meaning
348      * 100% of the balance is being spent.
349      * @return true if the full balance will be spent, false
350      * otherwise.
351      */
352     function isPayoutFullyDetermined() public view returns
353         (bool) {
354         uint256 sum = getDeterminedPayoutPercentage();
355         return sum == MAX_PERCENTAGE;
356     }
357
358     /**
359      * Calculates the percentage amount of currently
360      * determined payout.
361      * @return a number between 0 and 100, equivalent to the
362      * combined relative payout.
363      */
364
365     /**
366      * Returns the total number of active beneficiaries.
367      * @return the count of active beneficiaries.
368      */
369     function getActiveCount() public view returns
370         (uint256) {
371         return activeBeneficiaries.length;
372     }
373
374     /**
375      * Withdraws the specified amount from the Aave pool.
376      * @param amount The amount to withdraw.
377      */
378     function withdraw(uint amount) public {
379         aavePool.withdraw(amount);
380     }
381
382     /**
383      * Returns the current balance of the USDC token.
384      * @return the current balance.
385      */
386     function getBalance() public view returns
387         (uint256) {
388         return usdc.balanceOf(address(this));
389     }
390
391     /**
392      * Returns the percentage of the total balance that has been
393      * spent.
394      * @return the percentage.
395      */
396     function getDeterminedPayoutPercentage() public view returns
397         (uint256) {
398         uint256 totalBalance = getBalance();
399         uint256 spentBalance = aavePool.balanceOf(address(this));
400         if (totalBalance == 0) {
401             return 0;
402         }
403         return (spentBalance * 100) / totalBalance;
404     }
405
406     /**
407      * Sets the payout address for a specific beneficiary.
408      * @param index The index of the beneficiary.
409      * @param address The new payout address.
410      */
411     function setPayoutAddress(uint index, address address)
412         external {
413         activeBeneficiaries[index].payoutAddress = address;
414     }
415
416     /**
417      * Sets the amount for a specific beneficiary.
418      * @param index The index of the beneficiary.
419      * @param amount The new amount.
420      */
421     function setAmount(uint index, uint amount)
422         external {
423         activeBeneficiaries[index].amount = amount;
424     }
425
426     /**
427      * Sets the active status for a specific beneficiary.
428      * @param index The index of the beneficiary.
429      * @param active The new active status.
430      */
431     function setActiveStatus(uint index, bool active)
432         external {
433         activeBeneficiaries[index].active = active;
434     }
435
436     /**
437      * Sets the developer address.
438      * @param address The new developer address.
439      */
440     function setDeveloperAddress(address address)
441         external {
442         developerAddress = address;
443     }
444
445     /**
446      * Sets the payout percentage.
447      * @param percentage The new payout percentage.
448      */
449     function setPayoutPercentage(uint percentage)
450         external {
451         MAX_PERCENTAGE = percentage;
452     }
453
454     /**
455      * Sets the developer address.
456      * @param address The new developer address.
457      */
458     function setOurAddress(address address)
459         external {
460         ourAddress = address;
461     }
462
463     /**
464      * Sets the developer address.
465      * @param address The new developer address.
466      */
467     function setUsdcAddress(address address)
468         external {
469         usdcAddress = address;
470     }
471
472     /**
473      * Sets the developer address.
474      * @param address The new developer address.
475      */
476     function setAavePoolAddress(address address)
477         external {
478         aavePoolAddress = address;
479     }
480
481     /**
482      * Sets the developer address.
483      * @param address The new developer address.
484      */
485     function setHardhatAddress(address address)
486         external {
487         hardhatAddress = address;
488     }
489
490     /**
491      * Sets the developer address.
492      * @param address The new developer address.
493      */
494     function setPayoutAddress(address address)
495         external {
496         payoutAddress = address;
497     }
498
499     /**
500      * Sets the developer address.
501      * @param address The new developer address.
502      */
503     function setBeneficiaryAddress(address address)
504         external {
505         beneficiaryAddress = address;
506     }
507
508     /**
509      * Sets the developer address.
510      * @param address The new developer address.
511      */
512     function setContractAddress(address address)
513         external {
514         contractAddress = address;
515     }
516
517     /**
518      * Sets the developer address.
519      * @param address The new developer address.
520      */
521     function setTokenAddress(address address)
522         external {
523         tokenAddress = address;
524     }
525
526     /**
527      * Sets the developer address.
528      * @param address The new developer address.
529      */
530     function setPoolAddress(address address)
531         external {
532         poolAddress = address;
533     }
534
535     /**
536      * Sets the developer address.
537      * @param address The new developer address.
538      */
539     function setPoolAddress(address address)
540         external {
541         poolAddress = address;
542     }
543
544     /**
545      * Sets the developer address.
546      * @param address The new developer address.
547      */
548     function setPoolAddress(address address)
549         external {
550         poolAddress = address;
551     }
552
553     /**
554      * Sets the developer address.
555      * @param address The new developer address.
556      */
557     function setPoolAddress(address address)
558         external {
559         poolAddress = address;
560     }
561
562     /**
563      * Sets the developer address.
564      * @param address The new developer address.
565      */
566     function setPoolAddress(address address)
567         external {
568         poolAddress = address;
569     }
570
571     /**
572      * Sets the developer address.
573      * @param address The new developer address.
574      */
575     function setPoolAddress(address address)
576         external {
577         poolAddress = address;
578     }
579
580     /**
581      * Sets the developer address.
582      * @param address The new developer address.
583      */
584     function setPoolAddress(address address)
585         external {
586         poolAddress = address;
587     }
588
589     /**
590      * Sets the developer address.
591      * @param address The new developer address.
592      */
593     function setPoolAddress(address address)
594         external {
595         poolAddress = address;
596     }
597
598     /**
599      * Sets the developer address.
600      * @param address The new developer address.
601      */
602     function setPoolAddress(address address)
603         external {
604         poolAddress = address;
605     }
606
607     /**
608      * Sets the developer address.
609      * @param address The new developer address.
610      */
611     function setPoolAddress(address address)
612         external {
613         poolAddress = address;
614     }
615
616     /**
617      * Sets the developer address.
618      * @param address The new developer address.
619      */
620     function setPoolAddress(address address)
621         external {
622         poolAddress = address;
623     }
624
625     /**
626      * Sets the developer address.
627      * @param address The new developer address.
628      */
629     function setPoolAddress(address address)
630         external {
631         poolAddress = address;
632     }
633
634     /**
635      * Sets the developer address.
636      * @param address The new developer address.
637      */
638     function setPoolAddress(address address)
639         external {
640         poolAddress = address;
641     }
642
643     /**
644      * Sets the developer address.
645      * @param address The new developer address.
646      */
647     function setPoolAddress(address address)
648         external {
649         poolAddress = address;
650     }
651
652     /**
653      * Sets the developer address.
654      * @param address The new developer address.
655      */
656     function setPoolAddress(address address)
657         external {
658         poolAddress = address;
659     }
660
661     /**
662      * Sets the developer address.
663      * @param address The new developer address.
664      */
665     function setPoolAddress(address address)
666         external {
667         poolAddress = address;
668     }
669
670     /**
671      * Sets the developer address.
672      * @param address The new developer address.
673      */
674     function setPoolAddress(address address)
675         external {
676         poolAddress = address;
677     }
678
679     /**
680      * Sets the developer address.
681      * @param address The new developer address.
682      */
683     function setPoolAddress(address address)
684         external {
685         poolAddress = address;
686     }
687
688     /**
689      * Sets the developer address.
690      * @param address The new developer address.
691      */
692     function setPoolAddress(address address)
693         external {
694         poolAddress = address;
695     }
696
697     /**
698      * Sets the developer address.
699      * @param address The new developer address.
700      */
701     function setPoolAddress(address address)
702         external {
703         poolAddress = address;
704     }
705
706     /**
707      * Sets the developer address.
708      * @param address The new developer address.
709      */
710     function setPoolAddress(address address)
711         external {
712         poolAddress = address;
713     }
714
715     /**
716      * Sets the developer address.
717      * @param address The new developer address.
718      */
719     function setPoolAddress(address address)
720         external {
721         poolAddress = address;
722     }
723
724     /**
725      * Sets the developer address.
726      * @param address The new developer address.
727      */
728     function setPoolAddress(address address)
729         external {
730         poolAddress = address;
731     }
732
733     /**
734      * Sets the developer address.
735      * @param address The new developer address.
736      */
737     function setPoolAddress(address address)
738         external {
739         poolAddress = address;
740     }
741
742     /**
743      * Sets the developer address.
744      * @param address The new developer address.
745      */
746     function setPoolAddress(address address)
747         external {
748         poolAddress = address;
749     }
750
751     /**
752      * Sets the developer address.
753      * @param address The new developer address.
754      */
755     function setPoolAddress(address address)
756         external {
757         poolAddress = address;
758     }
759
760     /**
761      * Sets the developer address.
762      * @param address The new developer address.
763      */
764     function setPoolAddress(address address)
765         external {
766         poolAddress = address;
767     }
768
769     /**
770      * Sets the developer address.
771      * @param address The new developer address.
772      */
773     function setPoolAddress(address address)
774         external {
775         poolAddress = address;
776     }
777
778     /**
779      * Sets the developer address.
780      * @param address The new developer address.
781      */
782     function setPoolAddress(address address)
783         external {
784         poolAddress = address;
785     }
786
787     /**
788      * Sets the developer address.
789      * @param address The new developer address.
790      */
791     function setPoolAddress(address address)
792         external {
793         poolAddress = address;
794     }
795
796     /**
797      * Sets the developer address.
798      * @param address The new developer address.
799      */
800     function setPoolAddress(address address)
801         external {
802         poolAddress = address;
803     }
804
805     /**
806      * Sets the developer address.
807      * @param address The new developer address.
808      */
809     function setPoolAddress(address address)
810         external {
811         poolAddress = address;
812     }
813
814     /**
815      * Sets the developer address.
816      * @param address The new developer address.
817      */
818     function setPoolAddress(address address)
819         external {
820         poolAddress = address;
821     }
822
823     /**
824      * Sets the developer address.
825      * @param address The new developer address.
826      */
827     function setPoolAddress(address address)
828         external {
829         poolAddress = address;
830     }
831
832     /**
833      * Sets the developer address.
834      * @param address The new developer address.
835      */
836     function setPoolAddress(address address)
837         external {
838         poolAddress = address;
839     }
840
841     /**
842      * Sets the developer address.
843      * @param address The new developer address.
844      */
845     function setPoolAddress(address address)
846         external {
847         poolAddress = address;
848     }
849
850     /**
851      * Sets the developer address.
852      * @param address The new developer address.
853      */
854     function setPoolAddress(address address)
855         external {
856         poolAddress = address;
857     }
858
859     /**
860      * Sets the developer address.
861      * @param address The new developer address.
862      */
863     function setPoolAddress(address address)
864         external {
865         poolAddress = address;
866     }
867
868     /**
869      * Sets the developer address.
870      * @param address The new developer address.
871      */
872     function setPoolAddress(address address)
873         external {
874         poolAddress = address;
875     }
876
877     /**
878      * Sets the developer address.
879      * @param address The new developer address.
880      */
881     function setPoolAddress(address address)
882         external {
883         poolAddress = address;
884     }
885
886     /**
887      * Sets the developer address.
888      * @param address The new developer address.
889      */
890     function setPoolAddress(address address)
891         external {
892         poolAddress = address;
893     }
894
895     /**
896      * Sets the developer address.
897      * @param address The new developer address.
898      */
899     function setPoolAddress(address address)
900         external {
901         poolAddress = address;
902     }
903
904     /**
905      * Sets the developer address.
906      * @param address The new developer address.
907      */
908     function setPoolAddress(address address)
909         external {
910         poolAddress = address;
911     }
912
913     /**
914      * Sets the developer address.
915      * @param address The new developer address.
916      */
917     function setPoolAddress(address address)
918         external {
919         poolAddress = address;
920     }
921
922     /**
923      * Sets the developer address.
924      * @param address The new developer address.
925      */
926     function setPoolAddress(address address)
927         external {
928         poolAddress = address;
929     }
930
931     /**
932      * Sets the developer address.
933      * @param address The new developer address.
934      */
935     function setPoolAddress(address address)
936         external {
937         poolAddress = address;
938     }
939
940     /**
941      * Sets the developer address.
942      * @param address The new developer address.
943      */
944     function setPoolAddress(address address)
945         external {
946         poolAddress = address;
947     }
948
949     /**
950      * Sets the developer address.
951      * @param address The new developer address.
952      */
953     function setPoolAddress(address address)
954         external {
955         poolAddress = address;
956     }
957
958     /**
959      * Sets the developer address.
960      * @param address The new developer address.
961      */
962     function setPoolAddress(address address)
963         external {
964         poolAddress = address;
965     }
966
967     /**
968      * Sets the developer address.
969      * @param address The new developer address.
970      */
971     function setPoolAddress(address address)
972         external {
973         poolAddress = address;
974     }
975
976     /**
977      * Sets the developer address.
978      * @param address The new developer address.
979      */
980     function setPoolAddress(address address)
981         external {
982         poolAddress = address;
983     }
984
985     /**
986      * Sets the developer address.
987      * @param address The new developer address.
988      */
989     function setPoolAddress(address address)
990         external {
991         poolAddress = address;
992     }
993
994     /**
995      * Sets the developer address.
996      * @param address The new developer address.
997      */
998     function setPoolAddress(address address)
999         external {
1000        poolAddress = address;
1001    }
1002
1003    /**
1004      * Sets the developer address.
1005      * @param address The new developer address.
1006      */
1007      function setPoolAddress(address address)
1008          external {
1009          poolAddress = address;
1010      }
1011
1012    /**
1013      * Sets the developer address.
1014      * @param address The new developer address.
1015      */
1016      function setPoolAddress(address address)
1017          external {
1018          poolAddress = address;
1019      }
1020
1021    /**
1022      * Sets the developer address.
1023      * @param address The new developer address.
1024      */
1025      function setPoolAddress(address address)
1026          external {
1027          poolAddress = address;
1028      }
1029
1030    /**
1031      * Sets the developer address.
1032      * @param address The new developer address.
1033      */
1034      function setPoolAddress(address address)
1035          external {
1036          poolAddress = address;
1037      }
1038
1039    /**
1040      * Sets the developer address.
1041      * @param address The new developer address.
1042      */
1043      function setPoolAddress(address address)
1044          external {
1045          poolAddress = address;
1046      }
1047
1048    /**
1049      * Sets the developer address.
1050      * @param address The new developer address.
1051      */
1052      function setPoolAddress(address address)
1053          external {
1054          poolAddress = address;
1055      }
1056
1057    /**
1058      * Sets the developer address.
1059      * @param address The new developer address.
1060      */
1061      function setPoolAddress(address address)
1062          external {
1063          poolAddress = address;
1064      }
1065
1066    /**
1067      * Sets the developer address.
1068      * @param address The new developer address.
1069      */
1070      function setPoolAddress(address address)
1071          external {
1072          poolAddress = address;
1073      }
1074
1075    /**
1076      * Sets the developer address.
1077      * @param address The new developer address.
1078      */
1079      function setPoolAddress(address address)
1080          external {
1081          poolAddress = address;
1082      }
1083
1084    /**
1085      * Sets the developer address.
1086      * @param address The new developer address.
1087      */
1088      function setPoolAddress(address address)
1089          external {
1090          poolAddress = address;
1091      }
1092
1093    /**
1094      * Sets the developer address.
1095      * @param address The new developer address.
1096      */
1097      function setPoolAddress(address address)
1098          external {
1099          poolAddress = address;
1100      }
1101
1102    /**
1103      * Sets the developer address.
1104      * @param address The new developer address.
1105      */
1106      function setPoolAddress(address address)
1107          external {
1108          poolAddress = address;
1109      }
1110
1111    /**
1112      * Sets the developer address.
1113      * @param address The new developer address.
1114      */
1115      function setPoolAddress(address address)
1116          external {
1117          poolAddress = address;
1118      }
1119
1120    /**
1121      * Sets the developer address.
1122      * @param address The new developer address.
1123      */
1124      function setPoolAddress(address address)
1125          external {
1126          poolAddress = address;
1127      }
1128
1129    /**
1130      * Sets the developer address.
1131      * @param address The new developer address.
1132      */
1133      function setPoolAddress(address address)
1134          external {
1135          poolAddress = address;
1136      }
1137
1138    /**
1139      * Sets the developer address.
1140      * @param address The new developer address.
1141      */
1142      function setPoolAddress(address address)
1143          external {
1144          poolAddress = address;
1145      }
1146
1147    /**
1148      * Sets the developer address.
1149      * @param address The new developer address.
1150      */
1151      function setPoolAddress(address address)
1152          external {
1153          poolAddress = address;
1154      }
1155
1156    /**
1157      * Sets the developer address.
1158      * @param address The new developer address.
1159      */
1160      function setPoolAddress(address address)
1161          external {
1162          poolAddress = address;
1163      }
1164
1165    /**
1166      * Sets the developer address.
1167      * @param address The new developer address.
1168      */
1169      function setPoolAddress(address address)
1170          external {
1171          poolAddress = address;
1172      }
1173
1174    /**
1175      * Sets the developer address.
1176      * @param address The new developer address.
1177      */
1178      function setPoolAddress(address address)
1179          external {
1180          poolAddress = address;
1181      }
1182
1183    /**
1184      * Sets the developer address.
1185      * @param address The new developer address.
1186      */
1187      function setPoolAddress(address address)
1188          external {
1189          poolAddress = address;
1190      }
1191
1192    /**
1193      * Sets the developer address.
1194      * @param address The new developer address.
1195      */
1196      function setPoolAddress(address address)
1197          external {
1198          poolAddress = address;
1199      }
1200
1201    /**
1202      * Sets the developer address.
1203      * @param address The new developer address.
1204      */
1205      function setPoolAddress(address address)
1206          external {
1207          poolAddress = address;
1208      }
1209
1210    /**
1211      * Sets the developer address.
1212      * @param address The new developer address.
1213      */
1214      function setPoolAddress(address address)
1215          external {
1216          poolAddress = address;
1217      }
1218
1219    /**
1220      * Sets the developer address.
1221      * @param address The new developer address.
1222      */
1223      function setPoolAddress(address address)
1224          external {
1225          poolAddress = address;
1226      }
1227
1228    /**
1229      * Sets the developer address.
1230      * @param address The new developer address.
1231      */
1232      function setPoolAddress(address address)
1233          external {
1234          poolAddress = address;
1235      }
1236
1237    /**
1238      * Sets the developer address.
1239      * @param address The new developer address.
1240      */
1241      function setPoolAddress(address address)
1242          external {
1243          poolAddress = address;
1244      }
1245
1246    /**
1247      * Sets the developer address.
1248      * @param address The new developer address.
1249      */
1250      function setPoolAddress(address address)
1251          external {
1252          poolAddress = address;
1253      }
1254
1255    /**
1256      * Sets the developer address.
1257      * @param address The new developer address.
1258      */
1259      function setPoolAddress(address address)
1260          external {
1261          poolAddress = address;
1262      }
1263
1264    /**
1265      * Sets the developer address.
1266      * @param address The new developer address.
1267      */
1268      function setPoolAddress(address address)
1269          external {
1270          poolAddress = address;
1271      }
1272
1273    /**
1274      * Sets the developer address.
1275      * @param address The new developer address.
1276      */
1277      function setPoolAddress(address address)
1278          external {
1279          poolAddress = address;
1280      }
1281
1282    /**
1283      * Sets the developer address.
1284      * @param address The new developer address.
1285      */
1286      function setPoolAddress(address address)
1287          external {
1288          poolAddress = address;
1289      }
1290
1291    /**
1292      * Sets the developer address.
1293      * @param address The new developer address.
1294      */
1295      function setPoolAddress(address address)
1296          external {
1297          poolAddress = address;
1298      }
1299
1300    /**
1301      * Sets the developer address.
1302      * @param address The new developer address.
1303      */
1304      function setPoolAddress(address address)
1305          external {
1306          poolAddress = address;
1307      }
1308
1309    /**
1310      * Sets the developer address.
1311      * @param address The new developer address.
1312      */
1313      function setPoolAddress(address address)
1314          external {
1315          poolAddress = address;
1316      }
1317
1318    /**
1319      * Sets the developer address.
1320      * @param address The new developer address.
1321      */
1322      function setPoolAddress(address address)
1323          external {
1324          poolAddress = address;
1325      }
1326
1327    /**
1328      * Sets the developer address.
1329      * @param address The new developer address.
1330      */
1331      function setPoolAddress(address address)
1332          external {
1333          poolAddress = address;
1334      }
1335
1336    /**
1337      * Sets the developer address.
1338      * @param address The new developer address.
1339      */
1340      function setPoolAddress(address address)
1341          external {
1342          poolAddress = address;
1343      }
1344
1345    /**
1346      * Sets the developer address.
1347      * @param address The new developer address.
1348      */
1349      function setPoolAddress(address address)
1350          external {
1351          poolAddress = address;
1352      }
1353
1354    /**
1355      * Sets the developer address.
1356      * @param address The new developer address.
1357      */
1358      function setPoolAddress(address address)
1359          external {
1360          poolAddress = address;
1361      }
1362
1363    /**
1364      * Sets the developer address.
1365      * @param address The new developer address.
1366      */
1367      function setPoolAddress(address address)
1368          external {
1369          poolAddress = address;
1370      }
1371
1372    /**
1373      * Sets the developer address.
1374      * @param address The new developer address.
1375      */
1376      function setPoolAddress(address address)
1377          external {
1378          poolAddress = address;
1379      }
1380
1381    /**
1382      * Sets the developer address.
1383      * @param address The new developer address.
1384      */
1385      function setPoolAddress(address address)
1386          external {
1387          poolAddress = address;
1388      }
1389
1390    /**
1391      * Sets the developer address.
1392      * @param address The new developer address.
1393      */
1394      function setPoolAddress(address address)
1395          external {
1396          poolAddress = address;
1397      }
1398
1399    /**
1400      * Sets the developer address.
1401      * @param address The new developer address.
1402      */
1403      function setPoolAddress(address address)
1404          external {
1405          poolAddress = address;
1406      }
1407
1408    /**
1409      * Sets the developer address.
1410      * @param address The new developer address.
1411      */
1412      function setPoolAddress(address address)
1413          external {
1414          poolAddress = address;
1415      }
1416
1417    /**
1418      * Sets the developer address.
1419      * @param address The new developer address.
1420      */
1421      function setPoolAddress(address address)
1422          external {
1423          poolAddress = address;
1424      }
1425
1426    /**
1427      * Sets the developer address.
1428      * @param address The new developer address.
1429      */
1430      function setPoolAddress(address address)
1431          external {
1432          poolAddress = address;
1433      }
1434
1435    /**
1436      * Sets the developer address.
1437      * @param address The new developer address.
1438      */
1439      function setPoolAddress(address address)
1440          external {
1441          poolAddress = address;
1442      }
1443
1444    /**
1445      * Sets the developer address.
1446      * @param address The new developer address.
1447      */
1448      function setPoolAddress(address address)
1449          external {
1450          poolAddress = address;
1451      }
1452
1453    /**
1454      * Sets the developer address.
1455      * @param address The new developer address.
1456      */
1457      function setPoolAddress(address address)
1458          external {
1459          poolAddress = address;
1460      }
1461
1462    /**
1463      * Sets the developer address.
1464      * @param address The new developer address.
1465      */
1466      function setPoolAddress(address address)
1467          external {
1468          poolAddress = address;
1469      }
1470
1471    /**
1472      * Sets the developer address.
1473      * @param address The new developer address.
1474      */
1475      function setPoolAddress(address address)
1476          external {
1477          poolAddress = address;
1478      }
1479
1480    /**
1481      * Sets the developer address.
1482      * @param address The new developer address.
1483      */
1484      function setPoolAddress(address address)
1485          external {
1486          poolAddress = address;
1487      }
1488
1489    /**
1490      * Sets the developer address.
1491      * @param address The new developer address.
1492      */
1493      function setPoolAddress(address address)
1494          external {
1495          poolAddress = address;
1496      }
1497
1498    /**
1499      * Sets the developer address.
1500      * @param address The new developer address.
1501      */
1502      function setPoolAddress(address address)
1503          external {
1504          poolAddress = address;
1505      }
1506
1507    /**
1508      * Sets the developer address.
1509      * @param address The new developer address.
1510      */
1511      function setPoolAddress(address address)
1512          external {
1513          poolAddress = address;
1514      }
1515
1516    /**
1517      * Sets the developer address.
1518      * @param address The new developer address.
1519      */
1520      function setPoolAddress(address address)
1521          external {
1522          poolAddress = address;
1523      }
1524
1525    /**
1526      * Sets the developer address.
1527      * @param address The new developer address.
1528      */
1529      function setPoolAddress(address address)
1530          external {
1531          poolAddress = address;
1532      }
1533
1534    /**
1535      * Sets the developer address.
1536      * @param address The new developer address.
1537      */
1538      function setPoolAddress(address address)
1539          external {
1540          poolAddress = address;
1541      }
1542
1543    /**
1544      * Sets the developer address.
1545      * @param address The new developer address.
1546      */
1547      function setPoolAddress(address address)
1548          external {
1549          poolAddress = address;
1550      }
1551
1552    /**
1553      * Sets the developer address.
1554      * @param address The new developer address.
1555      */
1556      function setPoolAddress(address address)
1557          external {
1558          poolAddress = address;
1559      }
1560
1561    /**
1562      * Sets the developer address.
1563      * @param address The new developer address.
1564      */
1565      function setPoolAddress(address address)
1566          external {
1567          poolAddress = address;
1568      }
1569
1570    /**
1571      * Sets the developer address.
1572      * @param address The new developer address.
1573      */
1574      function setPoolAddress(address address)
1575          external {
1576          poolAddress = address;
1577      }
1578
1579    /**
1580      * Sets the developer address.
1581      * @param address The new developer address.
1582      */
1583      function setPoolAddress(address address)
1584          external {
1585          poolAddress = address;
1586      }
1587
1588    /**
1589      * Sets the developer address.
1590      * @param address The new developer address.
1591      */
1592      function setPoolAddress(address address)
1593          external {
1594          poolAddress = address;
1595      }
1596
1597    /**
1598      * Sets the developer address.
1599      * @param address The new developer address.
1600      */
1601      function setPoolAddress(address address)
1602          external {
1603          poolAddress = address;
1604      }
1605
1606    /**
1607      * Sets the developer address.
1608      * @param address The new developer address.
1609      */
1610      function setPoolAddress(address address)
1611          external {
1612          poolAddress = address;
1613      }
1614
1615    /**
1616      * Sets the developer address.
1617      * @param address The new developer address.
1618      */
1619      function setPoolAddress(address address)
1620          external {
1621          poolAddress = address;
1622      }
1623
1624    /**
1625      * Sets the developer address.
1626      * @param address The new developer address.
1627      */
1628      function setPoolAddress(address address)
1629          external {
1630          poolAddress = address;
1631      }
1632
1633    /**
1634      * Sets the developer address.
1635      * @param address The new developer address.
1636      */
1637      function setPoolAddress(address address)
1638          external {
1639          poolAddress = address;
1640      }
1641
1642    /**
1643      * Sets the developer address.
1644      * @param address The new developer address.
1645      */
1646      function setPoolAddress(address address)
1647          external {
1648          poolAddress = address;
1649      }
1650
1651    /**
1652      * Sets the developer address.
1653      * @param address The new developer address.
16
```

```

352     function getDeterminedPayoutPercentage() public view
353     returns (uint256) {
354         uint256 sum;
355         for (uint256 i = 0; i < MAX_BENEFICIARIES; i++) {
356             if (_beneficiaries[i].payoutAddress != address(0))
357             {
358                 sum += _beneficiaries[i].amount;
359             }
360         }
361     }
362     /**
363      * Gets the current balance.
364      * @return the balance of the combined deposited funds.
365      */
366     function getBalance() public view returns (uint256) {
367         return aavePool.getBalance(address(this));
368     }
369
370     /**
371      * Getter for the beneficiaries list.
372      * @return the list of 10 beneficiaries (might contain
373      * empty slots).
374      */
375     function getBeneficiaries() public view returns
376         (Beneficiary[10] memory) {
377         return _beneficiaries;
378     }
379
380     /**
381      * Counts the number of active beneficiaries.
382      * @return the number of active beneficiaries.
383      */
384     function getActiveCount() public view returns (uint256) {
385         uint256 count;
386         for (uint256 i = 0; i < MAX_BENEFICIARIES; i++) {
387             if (_beneficiaries[i].payoutAddress != address(0))
388             {
389                 count++;
390             }
391         }
392     }
393     /**
394      * Gets only the active beneficiaries.
395      * @return an array of beneficiaries.
396      */
397     function getActiveBeneficiaries() public view returns
398         (Beneficiary[] memory) {
399         uint256 activeCount = getActiveCount();
400         Beneficiary[] memory active = new
401             Beneficiary[](activeCount);
402         uint256 count = 0;
403         for (uint256 i = 0; i < MAX_BENEFICIARIES; i++) {
404             if (_beneficiaries[i].payoutAddress != address(0))
405             {

```

```

402             active[count] = _beneficiaries[i];
403             count++;
404         }
405     }
406     return active;
407 }
408
409 /**
410 * Gets the current state of the contract.
411 * @return the current state.
412 */
413 function getState() public view returns (State) {
414     return _currentState;
415 }
416
417 /**
418 * Gets the last check-in time.
419 * @return the last check-in time.
420 */
421 function getLastCheckIn() public view returns (uint256) {
422     return _lastCheckIn;
423 }
424
425 }
```

Listing 1: smart contract

## A References

### References

- [1] Bidget. *How Many Bitcoin Have Been Lost?* Accessed 2025-11-06. 2025. URL: <https://www.bitget.com/wiki/how-many-bitcoin-have-been-lost>.