UNIVERSITÀ DEGLI STUDI DI BERGAMO

SCUOLA DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Classe n.

Titolo

Relatore: Chiar.mo Prof./Chiar.ma Prof.ssa

Prova finale di Andrea Cattaneo
NOME COGNOME

Matricola n. 1040655

ANNO ACCADEMICO
2017 /2018

All'amato me stesso

Citatemi dicendo che sono stato citato male.

Groucho Marx

Ringraziamenti

Grazie a Grazia, Graziella e la sorella.

Sommario

Piacere, sommario.

Indice

Ringraziamenti III

Sommario IV

1 Introduzione 1

1.1 DLT 1

1.2 Blockchain 2

1.3 Smart Contracts 3

1.4 Stellar 3

1.4.1 Gli Smart Contracts in Stellar 3

2 Time-Lock Encryption 5

2.1 Cosa è la TLE 5

2.2 Requisiti 5

2.2.1 Certezza di pubblicazione del messaggio al tempo τ 5

2.2.2 Segretezza del messaggio sino al tempo τ 5

2.3 Come implementare la TLE 5

2.3.1 Time-Lock Puzzle 6

3 Il Protocollo 7

3.1 Versione base 7

3.2 Versione avanzata 11

4 Proof of Concept 15

4.1 Un'implementazione sullo Stellar Network 15

4.2 Limitazioni della soluzione proposta 18

4.2.1 Numero massimo di firme per transazione 18

4.2.2 Riscatto dei *counterprize* 18

4.3 Lo script 20

5 Scelta dei parametri del protocollo 28

5.1 I Parametri 28

5.1.1 La deadline τ 28

5.1.2 La tolleranza δ 28

5.1.3 Il *prize* 29

5.1.4 Il *counterprize* 29

5.1.5 Il *pawn* 29

5.1.6 Numero di share 30

5.1.7 Threshold per la ricostruzione del segreto 30

5.2 Criteri di dimensionamento 30

5.2.1 Denaro necessario 30

5.2.2 Costo 31

5.2.3 Resistenza a smarrimenti 31

5.2.4 Resistenza a furti 32

6 Analisi delle criticità 33

6.1 Comportamenti negligenti dei singoli utenti 34

6.1.1 Uno o più provider perdono il segreto 34

6.1.2 Il provider si fa rubare il segreto 34

6.1.3 Il client si fa rubare/pubblica il segreto prima di τ 34

6.2 Comportamenti malevoli dei singoli utenti 34
6.2.1 Comportamento malevole del client 34
6.2.2 Comportamento malevolo di un provider 34
6.3 Coalizioni tra provider 34
6.4 Un singolo attore controlla più provider 34
6.5 Un soggetto corrompe i provider 35

7 Possibili usi 36
7.1 Gara d'appalto a buste chiuse 36

A An appendix 37

Riferimenti bibliografici 40

Elenco delle figure

3.1 Step 0 8
3.2 Step ok 9
3.3 Step leggi 9
3.4 Step anticipo 10
3.5 Leak 1 10
3.6 Leak 2 10
3.7 Avanzato - Inizializzazione 11
3.8 Leak 1 12
3.9 Leak 2 12
3.10 Step leggi 12
3.11 Step leggi 13
3.12 Leak 1 13
3.13 Leak 2 13

Elenco delle tabelle

Listings

code/poc/main.py 20

Capitolo 1

Introduzione

La Blockchain è uno degli argomenti più caldi nell'industria dell'IT. In questo capitolo cercheremo di fare chiarezza, partendo dalle origini di questa tecnologia sino ad arrivare allo stato dell'arte odierno.

1.1 DLT

Per capire cosa è una **Distribuited Ledger Technology** (DLT) è necessario innanzitutto chiarire il concetto di Ledger.

Un **Ledger**, parola che in italiano traduciamo come *Libro Mastro*, è il registro principale in cui sono riunite tutte le transazioni economiche che compongono un dato sistema contabile. In sostanza si tratta di un documento nel quale vengono registrate tutte le transazioni economiche, dove viene indicata la data della transazione, la cifra scambiata e i soggetti coinvolti nella transazione (chi dà e chi riceve). Le modifiche apportate possono essere solo di tipo incrementale: non si possono apportare modifiche o cancellare record, ma solo aggiungere nuove righe al ledger.

Poiché di fatto contiene tutta la storia del sistema contabile, il legder è anche lo strumento da consultare quando si vuole stabilire quanto denaro possiede un certo

individuo, e se quindi può effettuare un pagamento oppure no per insufficienza di fondi.

Il Ledger è uno strumento che da secoli viene usato dall'uomo. Le implementazioni classiche si basano su un soggetto fidato, come ad esempio una banca, che è l'unico ad essere autorizzato ad apportare modifiche. In questo modo è

1.2 Blockchain

La **Blockchain** non è altro che una tipologia di DLT. Spesso nell'linguaggio comune queste due parole vengono usate come sinonimi, ma in questa tesi ci impegneremo nel mantenere questa distinzione.

Una Blockchain è una catena di transazioni in continua crescita. I transazioni sono raggruppati in blocchi, e questi blocchi sono collegati tra di loro con tecniche crittografiche.

Ogni blocco è composto da:

- L'hash crittografico del blocco precedente
- Un timestamp, ossia la data e l'ora dell'aggiunta del blocco alla catena
- Un insieme di transazioni

La Blockchain è stata inventata da Satoshi Nagamoto nel 2008 per essere utilizzata come ledger pubblico per la gestione della criptovaluta Bitcoin. Con la blockchain, per la prima volta si è potuto risolvere il problema del *double-spending* senza ricorrere ad una autorità fidata.

1.3 Smart Contracts

Uno **Smart Contract** è un protocollo computerizzato che ha lo scopo di facilitare, verificare o obbligare la negoziazione o l'esecuzione di un contratto. L'aspetto sicuramente più interessante è che permettono di garantire l'esecuzione di determinate azioni, come un pagamento, senza l'intervento di un soggetto terzo. Gli Smart Contracts furono pensati per la prima volta da Nick Szabo nel 1994, per poi essere formalizzati dal lui stesso nel 1997 [1].

1.4 Stellar

Stellar è un protocollo decentralizzato ed open source nato con lo scopo di poter trasferire denaro attraverso diversi paesi con costi di transazione irrisori. Il nome della criptovaluta scambiata è Lumen (**XLM**).

È stato creato nel 2014 da **Jed McCaleb** e **Joyce Kim**. Si tratta di un progetto fortemente ispirato da *Ripple* (di cui Jed McCaleb è co-fondatore). A differenza di quest'ultimo, Stellar si pone come obietivo il facilitare lo scambio di denaro tra privati invece che tra banche.

[...]

1.4.1 Gli Smart Contracts in Stellar

In Stellar gli Smart Contracts di Szabo sono implementati attraverso gli Stellar Smart Contracts.

Uno **Stellar Smart Contract** (SSC) è espresso come la composizione di transazioni connesse fra loro ed eseguite secondo certi vincoli. I vincoli che si possono utilizzare nella realizzazione di SSCs sono:

- *Multisignature (multifirma)* - Quali chiavi sono necessarie per autorizzare una certa transazione? Quali soggetti devono concordare in una certa circorstanza affinche si possano eseguire i passi?

La Multisignature è il concetto di richiedere firme di soggetti diversi per effettuare transazioni provenienti da un certo account. Attraverso pesi e soglie di firma, viene creata la rappresentazione del potere nelle firme.

- *Batching/Atomicity (batching/atomicità)* - Quali operazioni devono avvenire o tutte insieme o fallire? Cosa deve accadere per forzare il successo o il fallimento?

Il Batching è il concetto dell'includere più operazioni in un'unica transazione. L'atomicità è la garanzia che, data una serie di operazioni raggruppate in un'unica transazione, al momento dell'invio sul network se anche una sola operazione fallisce, tutte le operazioni nella transazione falliscono.

- *Sequence (sequenza)* - In che ordine deve essere elaborata una serie di transazioni? Quali sono le limitazioni e le dipendenze?

Il concetto di sequenza è rappresentato sullo Stellar network attraverso il numero di sequenza. Utilizzando i numeri di sequenza nella manipolazione delle transazioni è possibile garantire che transazioni specifiche non possano essere eseguite se viene inoltrata una transazione alternativa.

- *Time Bounds (limiti temporali)* - Quando è possibile elaborare una transazione?

I limiti di tempo sono vincoli sul periodo di tempo durante il quale una transazione è valida. L'utilizzo dei limiti di tempo consente di rappresentare i periodi di tempo in un SSC.

Capitolo 2

Time-Lock Encryption

2.1 Cosa è la TLE

La **Time-Lock Encryption** si pone come obiettivo l'invio di un messaggio nel futuro.

2.2 Requisiti

2.2.1 Certezza di pubblicazione del messaggio al tempo τ

2.2.2 Segretezza del messaggio sino al tempo τ

2.3 Come implementare la TLE

Sostanzialmente esistono due strategie per approcciare questo problema

2.3.1 Time-Lock Puzzle

L'idea è quella di crittografare il segreto attraverso un problema matematico, risolvibile con un calcolatore, che richiede un certo tempo τ stimabile per poter essere risolto. ... inserire algoritmo ... Le criticità di questo approccio sono diverse. In primo luogo vi è la difficoltà nel trovare un algoritmo che soddisfi i requisiti qua sopra, e soprattutto che sia robusto anche a fronte del miglioramento della potenza di calcolo dei computer del futuro.

Un'altra questione è necessario che vi sia qualcuno disposto a risolvere il problema

Capitolo 3

Il Protocollo

In questo capitolo verrà trattato il protocollo proposto per l'implementazione della Time-Lock Encryption sui DLT. Nella prima parte cercheremo di spiegare l'idea che sta alla base, anche con l'ausilio di un esempio. Nella seconda parte invece passeremo ad analizzare gli aspetti più formali, come le caratteristiche dello Smart Contract che serve implementare e i requisiti del DLT sui cui operare.

3.1 Versione base

Immaginiamo che Carol abbia bisogno di Time-Lock Encryption su un certo messaggio x. Per farlo decide farsi aiutare da Alice. Quest'ultima impegna a conservare il messaggio e ha renderlo pubblico solo dopo un certo istante di tempo τ. Diremo che Carol è il *client* e che Alice è il *provider*.

Alice però vuole essere retribuita per la conservazione di x. Viene quindi fissata una ricompensa *prize*. Inoltre Alice vuole essere sicura di ottenere la ricompensa se rispetta il suo impegno. Allo stesso tempo Carol vuole avere la certezza che Alice possa riscattare il premio solo se si comporta in maniera corretta. Carol inoltre non

vuole che soggetti terzi partecipino all'accordo, perché desidera che sia x sia l'accordo stesso rimangano segreti. Per ottenere ciò decidono di usare uno smart contract.

Inizializzazione Nella fase iniziale Carol cominuca il messaggio x ad Alice. Allo stesso tempo invia allo smart contract una certa cifra in criptomoneta che corrispone alla somma tra il *prize* da corrispondere ad Alice e un *pawn* che le verrà restituito al termine delle operazioni, un hash crittografico del messaggio x, l'istante di tempo τ e una tolleranza δ.

TODO aggiungere *fee*

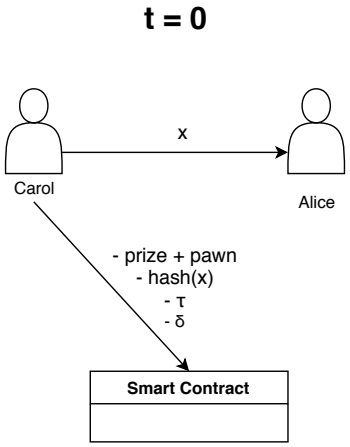

Figura 3.1: Step 0

Pubblicazione Alice conserva quindi x e lo rende noto al tempo t^* (con $\tau \leq t^* \leq \tau + \delta$).[1] Per farlo invia allo smart contract il messaggio x, ed in cambio ottiene il suo *prize*. Allo stesso tempo, Alice riottiene il *pawn* che aveva versato in precedenza. [2]

[1] Il δ serve a far si che Alice pubblichi puntualmente il messaggio. Se non ci fosse questo vincolo temporale potrebbe rilasciare il messaggio anche con molto ritardo rispetto a τ ed ottenere comunque la ricompensa.

[2] Ad una prima analisi può sembrare che il *pawn* sia inutile, ma in realtà è necessario per proteggersi da alcuni tipi di attacchi. Le ragioni dettagliate verranno discusse nel capitolo 6.

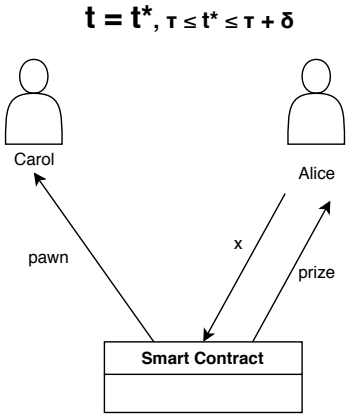

Figura 3.2: Step ok

Accesso al messaggio Dopo il tempo t^*, ossia dopo che Alice ha inviato x allo smart contract, il messaggio diventa pubblico e chiunque può leggerlo.

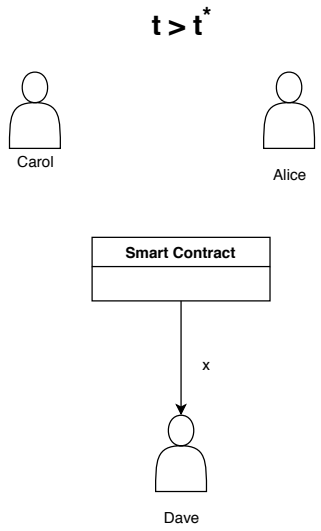

Figura 3.3: Step leggi

Pubblicazione anticipata Cosa succede se Alice prova a riscattare il premio prima dell'istante τ? Semplicemente lo smart contract rifiuta la sua richiesta.

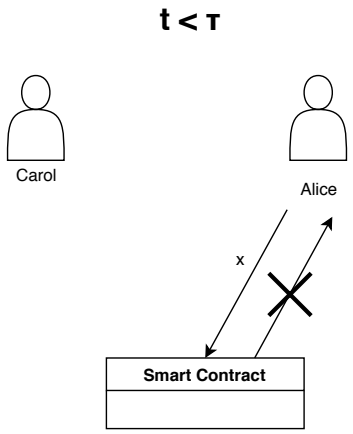

Figura 3.4: Step anticipo

Leak E se Alice cede (volontariamente o a causa di un furto) il segreto ad Eve prima del tempo τ? In questo caso Eve può usare x per ottenere una ricompensa *counterprize* [3] Il riscatto del *counterprize* impedisce ad Alice di ottenere il suo premio. È evidente che l'interesse di Alice è quello di mantenere x segreto.

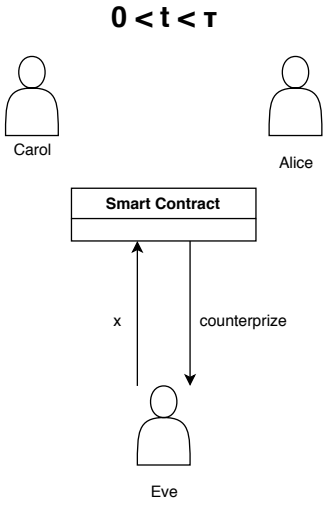

Figura 3.5: Leak 1

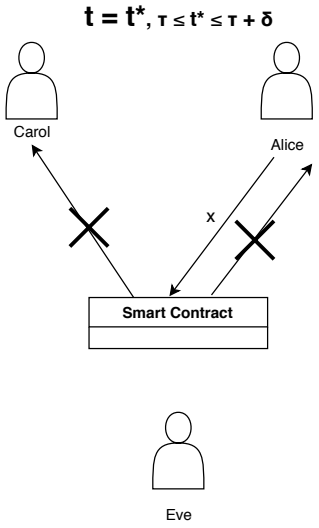

Figura 3.6: Leak 2

[3]dove *counterprize* $\ll$ *prize*. Le ragioni di questo vincolo sono spiegate al capitolo 6.

3.2 Versione avanzata

Facciamo notare che nella versione base (3.1) Alice conosce sin dall'inizio il messaggio x, perché le è stato affidato nella prima fase del processo. Ma se Carol volesse che il messaggio rimanga segreto anche ad Alice? Per soddisfare questa condizione Alice ha bisogno di (almeno) un altro *provider*, Bob, e di un algoritmo di **secret sharing**. [4]

Inizializzazione Alice, utilizzando un algoritmo di secret sharing, ottiene due share x_A e x_B. Invia questi share rispettivamente ad Alice e a Bob ed inizializza lo smart contract versando due *prize* e due *pawn*, inviando gli hash crittografici degli share, il tempo τ e la tolleranza δ.

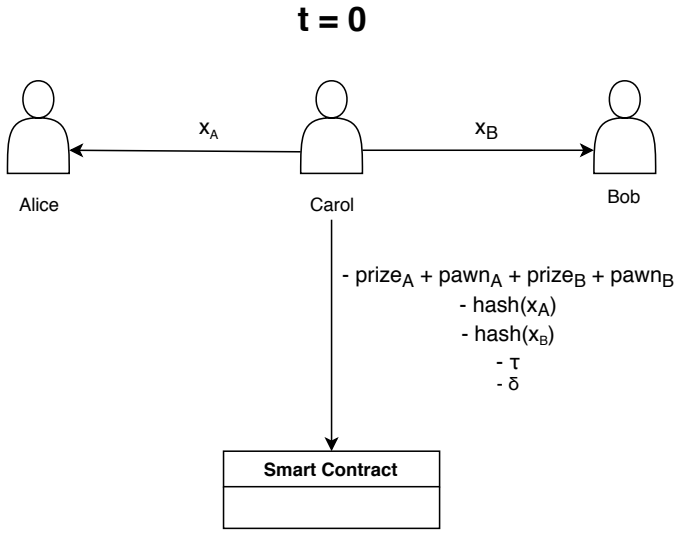

Figura 3.7: Avanzato - Inizializzazione

[4] Un algoritmo di secret sharing è un algoritmo che permette di distribuire un certo segreto tra un gruppo di partecipanti, ad ognuno dei quali viene assegnato uno *share*. Il segreto può essere ricostruito solo unendo un certo numero di share.

Pubblicazione Alice e Bob pubblicano i loro share rispettivamente al tempo t_A^* e t_B^* (con $\tau \leq t_A^* \leq \tau + \delta$, $\tau \leq t_B^* \leq \tau + \delta$). In cambio ottengono i loro *prize*. Allo stesso tempo Alice riottiene i rispettivi *pawn* che aveva versato in precedenza.

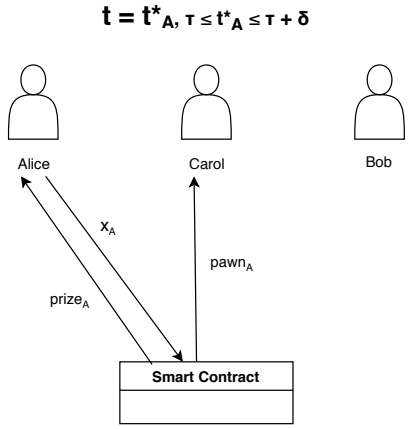

Figura 3.8: Leak 1

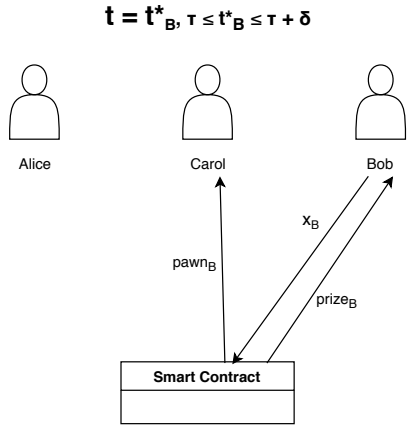

Figura 3.9: Leak 2

Accesso al messaggio Dopo il tempo $t^* = max(t_A^*, t_B^*)$, ossia dopo che sia Alice sia Bob hanno inviato x_A ed x_B allo smart contract, chiunque può leggere gli share dallo smart contract e quindi ricostruire il messaggio x.

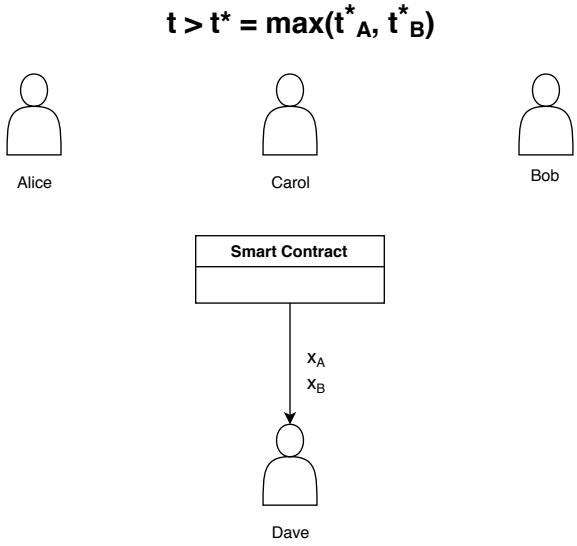

Figura 3.10: Step leggi

Leak Cosa succede se Bob cede il suo share ad Eve prima del tempo τ? Anche in questo caso Eve può usare x_B per ottenere una ricompensa *counterprize*. Il riscatto del *counterprize* impedirebbe a Bob di ottenere il suo premio. Da notare che se Alice si comporta in maniera corretta, ossia se tiene segreto il suo share, è in grado di ottenere la sua ricompensa indipendentemente dal comportamento di Bob.

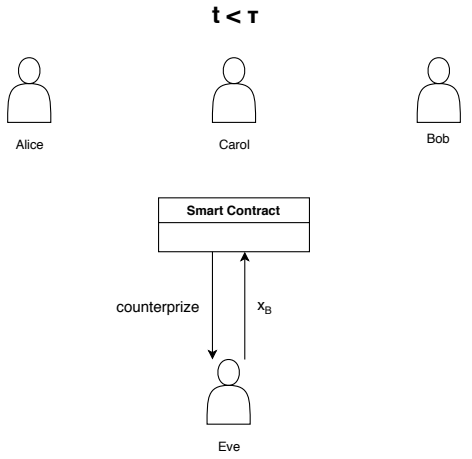

Figura 3.11: Step leggi

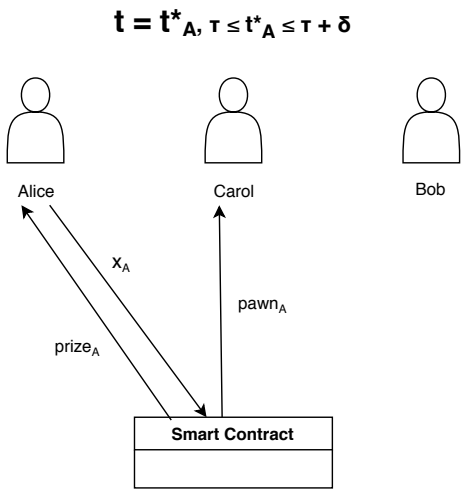

Figura 3.12: Leak 1

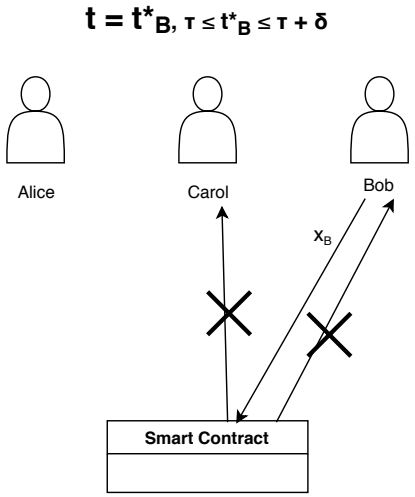

Figura 3.13: Leak 2

Abbiamo visto un esempio con la partecipazione di due *provider*, ma il protocollo

può essere analogamente implementato con N partecipanti, dove $N \geq 3$. Per una analisi approfondita su come sfruttare la ridondanza per garantire un buon grado di robustezza rimandiamo al capitolo 5.

Capitolo 4

Proof of Concept

In questo capitolo costruiremo una soluzione sullo Stellar Network compatibile con il protocollo proposto al capitolo 3. Seguirà una Proof of Concept di quanto descritto.

Avremmo potuto proporre una implementazione anche su altri DLT, come ad esempio Ethereum. Abbiamo scelto Stellar perché ci da modo di dimostrare che per un'implementazione (almeno parziale) del protocollo proposto non è necessario un ambiente che offre degli smart contract "potenti" come ad esempio della la Ethereum Virtual Machine, ma è sufficiente un supporto "debole" agli smart contract (1.4.1).

4.1 Un'implementazione sullo Stellar Network

Sia U_0 il *client*; siano $U_1, \dots, U_N$ i *provider* e siano rispettivamente $S_1, \dots, S_N$ i segreti assegnati ai provider (share o messaggi).

Per prima cosa U_0 crea gli account $A_1, \dots, A_N$ e versa su questi account una cifra pari alla somma tra il *prize*, il *pawn* e il costo che dovrà sostenere per le commissioni delle transazioni che dovrà effettuare. TODO SPIEGARE COME CALCOLARE IL COSTO DELLE COMMISSIONI.

A questo punto per ogni account A_i crea N transazioni ognuna delle quali rivolta ad uno degli utenti U_1, ... , U_N, per un totale di $N \times N$ transazioni. Indichiamo con $T_{A_iU_j}$ la transazione proveniente dall'account A_i e diretta all'utente U_j. Per semplicità di notazione poniamo $T_{A_iU_j} = T_{ij}$.

Le transazioni sono così fatte:

- T_{ii} **Transazione Prize**
  - SEQUENCE_NUMBER = nextSequenceNumber(A_i) + 1
  - MIN_TIME = τ
  - MAX_TIME = $\tau + \delta$
  - OPERATIONS
    1. SEND *prize* TO U_i
    2. SEND *pawn* TO U_0
- T_{ij}, con $i \neq j$ **Transazione Counterprize**
  - SEQUENCE_NUMBER = nextSequenceNumber(A_i) + 1
  - MIN_TIME = *None*
  - MAX_TIME = τ
  - OPERATIONS
    1. SEND *counterprize* TO U_j

Facciamo notare che tutte queste transazioni hanno lo stesso sequence number. Ciò significa che al più una di queste potrà essere eseguita.

Ora U_0 crea ulteriori N transazioni, una per ogni account, con queste caratteristiche:

- T_{i0} **Transazione d'inizializzazione**

- SEQUENCE_NUMBER = nextSequenceNumber(A_i)
- OPERATIONS
  1. SET *master weight* = 255
  2. SET *med threshold* = 2
  3. SET *high threshold* = 254
  4. APPEND PRE_AUTH_TX *hash(T_{ij})* WITH WEIGHT 1 (per ogni T_{ij}, con $1 \leq j \leq N$)
  5. APPEND HASHX_SIGNER *hash(S_i)* WITH WEIGHT 1
  6. SET *master weight* = 0

A U_0 non resta che inviare ad ogni utente U_i[1] il segreto S_i, le transazioni prize e counterprize T_{ij} e fare il submit delle transazioni d'inizializzazione T_{i0}.

Analizziamo nel dettaglio la transazione d'inizializzazione. Il *med threshold* è la soglia da raggiungere per effettuare l'invio di Lumen dall'account. Con l'operazione 2 lo abbiamo impostato a 2. Con le operazioni 4 abbiamo pre-autorizzato le transazioni T_{ij} con $1 \leq j \leq N$ con peso pari a 1. Con l'operazione 5 abbiamo aggiunto il segreto S_i tra le firme autorizzate, con peso pari a 1. Infine con le operazioni 3 e 6 abbiamo impedito all'utente U_0 (il detentore della masterkey) di effettuare alcun tipo di operazione dall'account A_i. L'operazione 1 ci è servita solo per non avere problemi nell'effettuare le operazioni successive.

Lo scenario finale è che l'account A_i è bloccato: neanche U_0, il detentore della master key può apportarvi modifiche o prelevare fondi. Le uniche operazioni autorizzate sono le T_{ij}, ossia le transazioni per ottenere il *prize* o il *counterprize*. Per fare il submit

[1] con $1 \leq i \leq N$

di queste transazioni è inoltre necessario conoscere S_i, perché in questo modo si riesce a raggiungere il med threshold.

4.2 Limitazioni della soluzione proposta

4.2.1 Numero massimo di firme per transazione

Una limitazione imposta dallo Stellar Network è che con una singola transazione è possibile aggiungere al massimo 20 firme ad un account. Ciò comporta che se $N >= 20$ è necessario creare più di una transazione d'inizializzazione per ogni account. In generale, il numero delle transazioni d'inizializzazione necessarie è

$$\#T_{i0} = \left\lfloor \frac{N+1}{20} \right\rfloor$$

4.2.2 Riscatto dei *counterprize*

L'implementazione proposta si basa sul fatto che in Stellar è possibile pre-autorizzare delle transazioni. Uno dei problemi è che non è possibile pre-autorizzare delle transazioni con un destinatario generico, ma è necessario definirlo a priori. Nel nostro caso ciò comporta che i possibili destinatari dei *counterprize* devono essere decisi già nella fase di inizializzazione. Per il tipo di soluzione proposta non vi è un limite teorico per il numero di transazioni *counterprize* che si possono emettere, ma chiaramente non è sostenibile l'emissione di transazioni verso tutti i potenziali account Stellar. Si è deciso quindi di emettere transazioni *counterprize* riscattabili solo dai provider coinvolti.

Ciò comporta che solo i provider possano riscattare il *counterprize*. Questo potrebbe essere un problema, perché comporta che un provider non avrebbe il disincentivo economico nel cedere il proprio share ad un soggetto esterno.

Eve ottiene lo share dal provider Alice. Eve a questo punto può decidere di dividersi il *counterprize* con il provider Bob. Per farlo Bob pre-autorizza due transazioni

- T_1
  - SEQUENCE_NUMBER = nextSequenceNumber(A_B) + 1
  - OPERATIONS
    1. SEND (*counterprize* / 2) TO U_j
    2. SET *master weight* = 1
- T_2
  - SEQUENCE_NUMBER = nextSequenceNumber(A_B) + 1
  - MIN_TIME = $\bar{t}$
  - MAX_TIME = *None*
  - OPERATIONS
    1. SET *master weight* = 1

e inizializza lo scambio con

- T_0
  - SEQUENCE_NUMBER = nextSequenceNumber(A_B)
  - OPERATIONS
    1. SEND *all* TO A'_B
    2. APPEND PRE_AUTH_TX *hash(T_1)* WITH WEIGHT 1
    3. APPEND PRE_AUTH_TX *hash(T_2)* WITH WEIGHT 1
    4. SET *master weight* = 0

Quello che avviene con la transazione T_0 e che Bob svuota l'account destinatario del *counterprize* A_B, si esclude dagli aventi diritto di firma su A_B e allo stesso tempo pre-autorizza le due transazioni T_1 e T_2.

La transazione T_1 invia ad Eve metà del *counterprize* e ripristina la firma di Bob su A_B. L'idea è che sia Eve a fare il submit di questa transazione. L'unico modo che ha per farlo è quello di fare il submit della transazione *counterprize*, in modo da avere sull'account A_B un deposito di fondi sufficienti.

Infine T_2 serve a tutelare Bob; se Eve non fa il submit di T_1 in un tempo ragionevole $\bar{t}$ Bob può usare T_2 per riprendere il possesso del suo account A_B.

4.3 Lo script

Di seguito uno script scritto in Python che applica quanto detto sopra. Si fa uso del *Python SDK per Stellar* e di *secretsharing di Blockstack*, un'implementazione open-source dello Shamir Secret Sharing.

Simuliamo la *versione avanzata*, con Carol che genera due share e li affida ad Alice e Bob. Simuliamo inoltre che Alice venga in possesso dello share di Bob, e quindi riesca ad ottenere il *counterprize* associato.

```
1 from stellar_base.keypair import Keypair
2 from stellar_base.address import Address
3 from stellar_base.builder import Builder
4 from stellar_base.transaction_envelope import
    TransactionEnvelope as Te
5 from stellar_base.stellarxdr.StellarXDR_type import
    TimeBounds
6 from stellar_base.horizon import horizon_testnet
7 from secretsharing import PlaintextToHexSecretSharer
8 import requests
```

```
9  import hashlib
10 import datetime
11 import time
12 import base64
13
14
15 def initialize_keypair(add_funds=True):
16     kp = Keypair.random()
17     if add_funds:
18         publickey = kp.address().decode()
19         url = 'https://friendbot.stellar.org/?addr=' +
              publickey
20         requests.get(url)
21     return kp
22
23
24 PRIZE = '100'
25 PAWN = '100'
26 COUNTERPRIZE = '1'
27 DELTA = 60  # seconds
28
29 x = 'Hello, World!'  # Carol's message
30 x_a, x_b = PlaintextToHexSecretSharer.split_secret(x, 2, 2)
31
32 hash_x_a = hashlib.sha256(x_a.encode()).digest()
33 hash_x_b = hashlib.sha256(x_b.encode()).digest()
34 horizon = horizon_testnet()
35
36 print('Initializing keypairs')
```

```
37 kp_alice = initialize_keypair()
38 kp_bob = initialize_keypair()
39 kp_carol = initialize_keypair()
40
41 print('Carol␣creates␣rho␣and␣gamma')
42 kp_rho = initialize_keypair(add_funds=False)
43 kp_gamma = initialize_keypair(add_funds=False)
44 kp_rho_address = kp_rho.address().decode()
45 kp_gamma_address = kp_gamma.address().decode()
46 kp_rho_seed = kp_rho.seed().decode()
47 kp_gamma_seed = kp_gamma.seed().decode()
48 builder_rho_gamma = Builder(secret=kp_carol.seed().decode())
49 builder_rho_gamma.append_create_account_op(kp_rho_address,
      '202.50009')
50 builder_rho_gamma.append_create_account_op(kp_gamma_address,
      '202.50009')
51 builder_rho_gamma.sign()
52 response = builder_rho_gamma.submit()
53 assert 'hash' in response
54
55 print('Initializing␣transactions')
56 address_rho = Address(address=kp_rho_address)
57 address_gamma = Address(address=kp_gamma_address)
58 address_rho.get()
59 address_gamma.get()
60
61 starting_sequence_rho = int(address_rho.sequence)
62 starting_sequence_gamma = int(address_gamma.sequence)
63
```

```
64 tau = datetime.datetime.now() +
      datetime.timedelta(seconds=30)
65 tau_unix = int(time.mktime(tau.timetuple()))
66 tau_plus_delta = tau + datetime.timedelta(seconds=DELTA)
67 tau_plus_delta_unix =
      int(time.mktime(tau_plus_delta.timetuple()))
68 timebound_transazione = TimeBounds(
69     minTime=tau_unix, maxTime=tau_plus_delta_unix)
70 timebound_controtransazione = TimeBounds(minTime=0,
      maxTime=tau_unix)
71
72 builder_rho_1 = Builder(secret=kp_rho_seed,
73                         sequence=starting_sequence_rho+1)
74 builder_rho_1.append_payment_op(kp_bob.address().decode(),
      COUNTERPRIZE, 'XLM')
75 builder_rho_1.add_time_bounds(timebound_controtransazione)
76 tx_rho_1 = builder_rho_1.gen_tx()
77 hash_rho_1 = builder_rho_1.gen_te().hash_meta()
78
79 builder_rho_2 = Builder(secret=kp_rho_seed,
80                         sequence=starting_sequence_rho+1)
81 builder_rho_2.append_payment_op(kp_alice.address().decode(),
      PRIZE, 'XLM')
82 builder_rho_2.append_payment_op(kp_carol.address().decode(),
      PAWN, 'XLM')
83 builder_rho_2.add_time_bounds(timebound_transazione)
84 tx_rho_2 = builder_rho_2.gen_tx()
85 hash_rho_2 = builder_rho_2.gen_te().hash_meta()
86
```

```
builder_gamma_1 = Builder(secret=kp_gamma_seed,
                          sequence=starting_sequence_gamma+1)
builder_gamma_1.append_payment_op(
    kp_alice.address().decode(), COUNTERPRIZE, 'XLM')
builder_gamma_1.add_time_bounds(timebound_controtransazione)
tx_gamma_1 = builder_gamma_1.gen_tx()
hash_gamma_1 = builder_gamma_1.gen_te().hash_meta()

builder_gamma_2 = Builder(secret=kp_gamma_seed,
                          sequence=starting_sequence_gamma+1)
builder_gamma_2.append_payment_op(kp_bob.address().decode(),
   PRIZE, 'XLM')
builder_gamma_2.append_payment_op(kp_carol.address().decode(),
   PAWN, 'XLM')
builder_gamma_2.add_time_bounds(timebound_transazione)
tx_gamma_2 = builder_gamma_2.gen_tx()
hash_gamma_2 = builder_gamma_2.gen_te().hash_meta()

builder_rho_0 = Builder(secret=kp_rho_seed)
builder_rho_0.append_set_options_op(master_weight=255)
builder_rho_0.append_set_options_op(med_threshold=2)
builder_rho_0.append_set_options_op(high_threshold=254)
builder_rho_0.append_pre_auth_tx_signer(hash_rho_1, 1)
builder_rho_0.append_pre_auth_tx_signer(hash_rho_2, 1)
builder_rho_0.append_hashx_signer(hash_x_a, 1)
builder_rho_0.append_set_options_op(master_weight=0)
builder_rho_0.sign()

builder_gamma_0 = Builder(secret=kp_gamma_seed)
```

```
114 builder_gamma_0.append_set_options_op(master_weight=255)
115 builder_gamma_0.append_set_options_op(med_threshold=2)
116 builder_gamma_0.append_set_options_op(high_threshold=254)
117 builder_gamma_0.append_pre_auth_tx_signer(hash_gamma_1, 1)
118 builder_gamma_0.append_pre_auth_tx_signer(hash_gamma_2, 1)
119 builder_gamma_0.append_hashx_signer(hash_x_b, 1)
120 builder_gamma_0.append_set_options_op(master_weight=0)
121 builder_gamma_0.sign()
122
123 print('Submitting␣rho_0')
124 response = builder_rho_0.submit()
125 assert 'hash' in response
126
127 print('Submitting␣gamma_0')
128 response = builder_gamma_0.submit()
129 assert 'hash' in response
130
131 print('At␣this␣point␣Carol␣cannot␣remove␣funds␣from␣
       rho/gamma')
132 builder = Builder(secret=kp_rho_seed)
133 builder.append_payment_op(kp_carol.address().decode(), 1)
134 builder.sign()
135 response = builder.submit()
136 assert 'hash' not in response
137
138 print('Alice␣tries␣to␣submit␣rho_2␣before␣the␣deadline,␣but␣
       fails')
139 print(f'[deadline:␣{␣tau␣};␣current␣time:␣{␣
       datetime.datetime.now()␣}]')
```

```
140 envelope = Te(tx_rho_2, opts={})
141 response = horizon.submit(envelope.xdr())
142 assert 'hash' not in response
143
144 print('Bob leaks his secret, so Alice can submit tx_gamma_1')
145 envelope = Te(tx_gamma_1, opts={})
146 envelope.sign_hashX(x_b)
147 response = horizon.submit(envelope.xdr())
148 assert 'hash' in response
149
150 print('Waiting for the deadline')
151 tts = (tau - datetime.datetime.now()).total_seconds()
152 time.sleep(tts)
153 time.sleep(5)  # some margin
154
155 print('Now Alice can submit tx_rho_2')
156 envelope = Te(tx_rho_2, opts={})
157 envelope.sign_hashX(x_a)
158 response = horizon.submit(envelope.xdr())
159 assert 'hash' in response
160
161 print('Bob cannot submit tx_gamma_2, because he leaked the
      secret')
162 envelope = Te(tx_gamma_2, opts={})
163 envelope.sign_hashX(x_b)
164 response = horizon.submit(envelope.xdr())
165 assert 'hash' not in response
166
167 print('At this point, the secret is public')
```

```
last_tx_rho = horizon.account_transactions(
    kp_rho_address, params={'limit': 1, 'order': 'desc'})
last_tx_gamma = horizon.account_transactions(
    kp_gamma_address, params={'limit': 1, 'order': 'desc'})
x_rho = base64.b64decode(
    last_tx_rho['_embedded']['records'][0]['signatures'][0]).decode()
x_gamma = base64.b64decode(
    last_tx_gamma['_embedded']['records'][0]['signatures'][0]).decode()
assert PlaintextToHexSecretSharer.recover_secret([x_rho,
   x_gamma]) == x
```

https://github.com/imcatta/stellar-time-lock-encryption-poc

Capitolo 5

Scelta dei parametri del protocollo

In questo capitolo vedremo quali sono i parametri del protocollo e analizzaremo alcuni criteri per il dimensionamento.

5.1 I Parametri

5.1.1 La deadline τ

La deadline τ rappresenta l'istante di tempo in cui si desidera che il segreto venga reso pubblico. Si tratta di un parametro deciso a priori, sul quale difficilmente si riesce ad intervenire. Ad ogni modo è bene sapere che più τ è lontano, più aumentano le possibilità che qualcosa vada storto.

5.1.2 La tolleranza δ

La tolleranza δ rappresenta quanto ritardo rispetto a τ è accettato per la pubblicazione del segreto. Una tolleranza troppo piccola potrebbe causare problemi se il client non è abbastanza reattivo al tempo τ, ad esempio per un intasamento del

network o perché è temporaneamente offline. Allo stesso tempo una tolleranza troppo grande potrebbe causare un ritardo non accettabile rispetto alla deadline. Per la scelta di questo parametro è ben tenere in considerazione anche la velocità di elaborazione delle transazioni del DLT sottostante.

5.1.3 Il *prize*

Il *prize* è la ricompensa da corrispondere al provider per il suo servizio di detenzione del segreto. È una cifra che deve essere concordata tra le parti, ed è bene che sia proporzionale a quanto dista τ dal momento dell'inizializzazione del protocollo e a quanto è importante il segreto.

$$prize \propto \tau - t_{init}$$

5.1.4 Il *counterprize*

Il *counterprize* è la cifra che viene corrisposta a chi invia il segreto allo smart contract prima del tempo τ. È bene che

$$prize \gg counterprize$$

Il motivo di questo vincolo è che il provider potrebbe in un qualsiasi istante prima di τ ottenere il *counterprize* (perché chiaramente conosce il segreto) e quindi rendere noto il segreto prima della deadline. Se però il *prize* è molto più grande sarebbe contro il suo interesse farlo, poiché perderebbe la possibilità di ottenere un guadagno significativamente maggiore al tempo τ.

5.1.5 Il *pawn*

Il *pawn* è la cifra che viene restituita al client quando il segreto viene pubblicato nella finestra temporale $\tau \leq t \leq \tau + \delta$.

Il *pawn* è necessario perché, ovviamente, anche il client conosce il segreto e quindi potrebbe ottenere il *counterprize* (magari poco prima della deadline). Se lo facesse il provider non potrebbe più ottenere il *prize* anche se si è comportato in maniera corretta, venendo in un certo senso truffato dal client. Come nel caso del *counterprize* si vuole scoraggiare questo comportamento con il vincolo

$$pawn \gg counterprize$$

5.1.6 Numero di share

Il numero di share n è uno dei parametri dell'algoritmo di secret sharing, utilizzato dalla versione avanzata del protocollo. La scelta di questo valore determina automaticamente anche il numero di provider necessari. Facciamo notare che la versione base può essere vista come un caso particolare della versione avanzata, dove $n = 1$.

5.1.7 Threshold per la ricostruzione del segreto

Il threshold t rappresenta il numero minimo di share che sono necessari per la ricostruzione del segreto. Anche in questo caso la versione base può essere vista come un caso particolare della versione avanzata, dove $t = 1$.

5.2 Criteri di dimensionamento

5.2.1 Denaro necessario

Il denaro necessario d rappresenta quanto denaro è necessario per inizializzare il protocollo.

$$d = \sum_{i=1}^{N} prize_i + pawn_i + fee_i$$

Nell'ipotesi che tutti i *prize* e tutti i *pawn* siano uguali tra loro e che le commissioni *fee* siano trascurabili l'espressione diventa

$$d = N(prize + pawn)$$

N.B. Questo valore non è il costo che deve sostenere il client, perché parte di questi soldi gli verranno restituiti attraverso i *pawn*.

5.2.2 Costo

Il costo c rappresenta quanto denaro dovrà spendere il client per il servizio richiesto. Il costo non è fissato a priori, perché dipende da quanti *pawn* verranno restituiti.

$$c = \sum_{i=1}^{N}(prize_i + fee_i) + \sum pawn_j$$

dove $\sum pawn_j$ è la somma dei *pawn* restituiti.

Nel caso migliore, ossia quando tutti i *pawn* ritornano al client diventa

$$c = \sum_{i=1}^{N}(prize_i + fee_i)$$

Mentre nel caso peggiore, ossia quando nessun *pawn* ritorna al client, è pari a

$$c = d = \sum_{i=1}^{N}(prize_i + pawn_i + fee_i)$$

5.2.3 Resistenza a smarrimenti

La resistenza agli smarrimenti θ rappresenta il numero di provider che possono non rendere noto il proprio share senza compromettere la pubblicazione del messaggio originale.

$$\theta = n - t$$

È interessante valutare questo valore in relazione al numero totale di share n.

$$\theta_r = \frac{n-t}{n}$$

$$\theta_{\%} = 100 \cdot \theta_r = 100 \cdot \frac{n-t}{n}$$

5.2.4 Resistenza a furti

La resistenza ai furti γ è il numero di share che un soggetto deve riuscire ad ottenere dai provider affinchè possa ricostruire il segreto prima del tempo τ. Questo numero è ovviamente pari al threshold.

$$\gamma = t$$

Capitolo 6

Analisi delle criticità

In questo capitolo analizzeremo le principali criticità a cui il protocollo proposto è esposto. Presenteremo i diversi scenari divisi per punti, ma è bene sapere che acune di queste situazioni potrebbero verificarsi anche allo stesso tempo.

6.1 Comportamenti negligenti dei singoli utenti

6.1.1 Uno o più provider perdono il segreto

6.1.2 Il provider si fa rubare il segreto

6.1.3 Il client si fa rubare/pubblica il segreto prima di τ

6.2 Comportamenti malevoli dei singoli utenti

6.2.1 Comportamento malevole del client

6.2.2 Comportamento malevolo di un provider

6.3 Coalizioni tra provider

6.4 Un singolo attore controlla più provider

DLT PERMISSIONED Affinché il protocollo funzioni al meglio è bene che tutti gli N provider siano entità ben distinte. Purtroppo non in tutti i contesti questa caratteristica è facile da verificare. Possono quindi venirsi a creare situazioni nelle quali un unico soggetto controlla più provider. Fissato k il numero di provider che controlla, individuiamo due casi:

Un singolo soggetto controlla $k < \gamma$ provider In questo caso il soggetto non dispone di un numero sufficiente di share per ricostruire il segreto. Bisogna considerare però che parte da una posizione di vantaggio, perché deve recuperare solo altri $\gamma - k$ share.

Un singolo soggetto controlla $k \geq \gamma$ provider In questo caso il soggetto dispone di un numero sufficiente di share per ricostruire il segreto. Significa che sin dal momento dell'inizializzazione del protocollo viene meno il requisito di segretezza del messaggio [vedi 2.2.2].

6.5 Un soggetto corrompe i provider

Un attaccante che vuole ottenere il messaggio prima del tempo τ deve riuscire ad ottenere almeno γ share.

Un tecnica che può usare è quella di "corrompere" un numero γ di provider. Per farlo deve offrire ad ogni provider una cifra maggiore di *prize*. Il provider ha interesse nel tenere il proprio share segreto perché sa che se lo fa al tempo τ può ottenere una ricompensa *prize*. Se però un soggetto gli offre una cifra maggiore o uguale di *prize* a quel punto l'azione più conviente diventa cedere il segreto.

Come è possibile difendersi? Quello che può fare il client è fissare dei *prize* adeguatamente alti rispetto al valore del segreto. Un'altra tecnica che può aiutare è quella di apportare un'ulteriore penalizzazione nel caso in cui il provider ceda il segreto. Queste penalizzazioni possono essere legate al particolare dominio applicativo in cui il protocollo viene utilizzato. Un esempio è visibile nell'ultimo capitolo.

Capitolo 7

Possibili usi

7.1 Gara d'appalto a buste chiuse

Appendice A

An appendix

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante

lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies

vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Bibliografia

[1] Nick Szabo. «Formalizing and Securing Relationships on Public Networks». In: *First Monday* 2.9 (1997). ISSN: 13960466. DOI: 10.5210/fm.v2i9.548. URL: http://ojphi.org/ojs/index.php/fm/article/view/548.