Preventing a Fork in a Blockchain – David Fighting Goliath

Przemysław Kubiak and Mirosław Kutyłowski

IEEE TrustCom 2020, Guangzhou

Looking for a Cheap Blockchain

Standard Blockchain

- huge energy consumption
- global scale multi-party solution

Looking for a Cheap Blockchain

Standard Blockchain

- huge energy consumption
- global scale multi-party solution

What about a small scale blockchain?

- run on a **small system**
- as **simple** as possible
- ... but still **provable** infeasibility of manipulations of this *append-only* data structure

⇒ ***a blockchain for "middle and small enterprises"?***

Dolev&Liber Blockchain System

Shlomi Dolev, Matan Liber, CSCML 2020, Beer Sheva, Israel: *Toward Self-stabilizing Blockchain, Reconstructing Totally Erased Blockchain*

erasure resilient blockchain

situation:

- a blockchain is managed by a small subgroup of the participants,

Dolev&Liber Blockchain System

Shlomi Dolev, Matan Liber, CSCML 2020, Beer Sheva, Israel: ***Toward Self-stabilizing Blockchain, Reconstructing Totally Erased Blockchain***

erasure resilient blockchain

situation:

- a blockchain is managed by a small subgroup of the participants,
- ... these blockchain managers might be attacked,
- ... and their copies of the blockchain are (at least partially) destroyed.

Dolev&Liber Blockchain System

Shlomi Dolev, Matan Liber, CSCML 2020, Beer Sheva, Israel: ***Toward Self-stabilizing Blockchain, Reconstructing Totally Erased Blockchain***

erasure resilient blockchain

situation:

- a blockchain is managed by a small subgroup of the participants,
- ... these blockchain managers might be attacked,
- ... and their copies of the blockchain are (at least partially) destroyed.

problems to be avoided:

- loosing transactions' history
- loosing account balances

Dolev&Liber Blockchain System

Restoration procedure

- **works even if a participant has lost or hides his own transaction history**
- local transaction copies of other users enable restoration

Dolev&Liber Blockchain System

Restoration procedure

- **works even if a participant has lost or hides his own transaction history**
- local transaction copies of other users enable restoration

a user $\mathcal{D}$ locally holds:

- own payments history – *payments linked lists* (PLL)
- own incomes history – *incomes linked lists* (ILL)

Dolev&Liber Blockchain System

Restoration procedure

- **works even if a participant has lost or hides his own transaction history**
- local transaction copies of other users enable restoration

a user $\mathcal{D}$ locally holds:

- own payments history – *payments linked lists* (PLL)
- own incomes history – *incomes linked lists* (ILL)

Problem: upon a restoration request $\mathcal{D}$ has incentive in:

- sending the complete ILL
- **hiding** at least a part of own PLL

Dolev&Liber Blockchain System

Trick to discourage a user $\mathcal{D}$ to submit an incomplete PLL

if $\mathcal{D}$ truncates own PLL, then the signing key ρ of $\mathcal{D}$ leaks:

- ρ encoded via verifiable secret sharing,
- a leakage by duplication of $\mathcal{D}$'s PLL entries on the ILLs of $\mathcal{D}$'s payees
  $\Rightarrow$ provides enough shares for secret reconstruction

Dolev&Liber Blockchain System

leakage penalty mechanism

$\mathcal{D}$ holds a pair $(\rho, y = g^{\rho})$ for DSA signatures

For transaction i:

- $\mathcal{D}$ generates a fresh polynomial

  $$Pol_i(x) = \rho + c_{i,1}x + c_{i,2}x^2,$$

  for $c_{i,1}, c_{i,2}$ chosen at random and kept secret by $\mathcal{D}$
- $\mathcal{D}$ must disclose $C_{i,1} = g^{c_{i,1}}$ and $C_{i,2} = g^{c_{i,2}}$.
- $\mathcal{D}$ generates shares: $Pol_i(1)$, $Pol_{i-1}(2)$,

Registering a request i for transaction data t_i:

$\mathcal{D}$ transfers to a permissioned node a pair $R_i = (T_i, D_i)$ where

$$\begin{aligned} T_i &= (t_i, \mathrm{DSA}_{\rho}(t_i)), \\ D_i &= (Pol_i(1), Pol_{i-1}(2), C_{i,1}, C_{i,2}, C_{i-1,1}, C_{i-1,2}) \end{aligned}$$

Dolev&Liber Blockchain System

leakage penalty mechanism

Checking consistency of the request

$Pol_i(1)$, $Pol_{i-1}(2)$ are consistent with the pairs $C_{i,1}$, $C_{i,2}$ and $C_{i-1,1}$, $C_{i-1,2}$, if:

$$\begin{aligned} g^{Pol_i(1)} &= (g^{\rho+c_{i,1}\cdot 1+c_{i,2}\cdot 1^2}) = y \cdot C_{i,1} \cdot C_{i,2} \\ g^{Pol_{i-1}(2)} &= (g^{\rho+c_{i-1,1}\cdot 2+c_{i-1,2}\cdot 2^2}) = y \cdot C^2_{i-1,1} \cdot C^4_{i-1,2} \end{aligned}$$

When the transaction is added to the blockchain the current transaction number of the user $\mathcal{D}$ is incremented to $i+1$.

Dolev&Liber Blockchain System

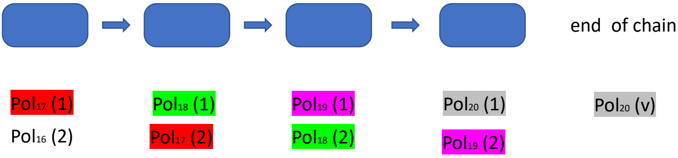
leakage penalty mechanism

Proving the end of PLL:

- let m be the index of the last transaction of $\mathcal{D}$
- since the request R_m has been processed, the share $s_m(1)$ has been revealed
- to prove that T_m was the last transaction, $\mathcal{D}$ must disclose

$$Pol_m(v) \bmod q),$$

for $v > 2$ chosen at random.



Pol_{17} (1)	Pol_{18} (1)	Pol_{19} (1)	Pol_{20} (1)	Pol_{20} (v)
Pol_{16} (2)	Pol_{17} (2)	Pol_{18} (2)	Pol_{19} (2)	

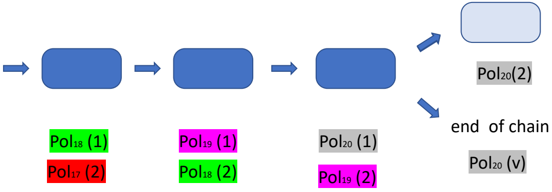
Dolev&Liber Blockchain System

leakage penalty mechanism

Proving the end of PLL:

- if PLL list has been truncated and m is not the number of the last transaction of $\mathcal{D}$, then
    - $Pol_m(2)$ already appeared somewhere
    - 3 shares $Pol_m(1)$, $Pol_m(2)$ and $Pol_m(v)$ of polynomial of degree 2 are available
    - ... this is enough for Lagrangian interpolation

$\Rightarrow$ one can derive the secret ρ



Pol_{18} (1)

Pol_{17} (2)

Pol_{19} (1)

Pol_{18} (2)

Pol_{20} (1)

Pol_{19} (2)

Pol_{20}(2)

end of chain

Pol_{20} (v)

Dolev&Liber Blockchain System

Does it suffice?

Yes, if the cryptographic assumptions are valid and **nobody** can break it.

David and Goliath Adversary Model

Goliath

a powerful adversary that **can break** the underlying cryptographic assumptions

David

a regular user for which breaking the cryptographic assumptions is **infeasible**

...maybe this reflects the current reality ...

Dolev&Liber Blockchain and Goliath

What all-mighty Goliath can achieve:

1. derive ρ and $c_{t,1}$ by computing discrete logarithm of y and $C_{t,1}$, respectively
2. solve the linear equation $Pol_t(1) = \rho + c_{t,1} + c_{t,2} \bmod q$ with a single unknown $c_{t,2}$
3. learn the polynomial Pol_t and calculate any share $Pol_t(j)$

$\Rightarrow$ **Goliath can impersonate David at any stage:**

- sign contracts on behalf of David
- **create a proof of David's misconduct**

Blockchain

David and Goliath model

Goals:

while it is impossible to prevent Goliath from breaking cryptographic assumptions the goal is that

> **David can defend himself by proving that somebody has broken the assumptions**

Consequence

- **impersonation finally fails**
- **proof of misconduct finally fails**

the only thing that Goliath can really achieve is to destroy the blockchain

... but this is always possible with a physical attack

The Failstop Signatures Based on DLP

E. van Heyst and T. P. Pedersen, *How to make efficient fail-stop signatures*, EUROCRYPT'92

Failstop mechanism

- let $h \in \langle g \rangle$ be such that David cannot know $\log_g(h)$ (of course, Goliath knows $\log_g(h)$)
- a one-time secret key of David is

$$SK = (x_1, x_2, y_1, y_2)$$

- the corresponding public key of David is

$$PK = (p_1, p_2) = (g^{x_1} \cdot h^{x_2}, g^{y_1} \cdot h^{y_2})$$

The Failstop Signatures Based on DLP

Signing *m* by David

$$\mathrm{Sign}(SK, m) := (\sigma_1, \sigma_2)$$

where

$$\begin{aligned}\sigma_1 &:= x_1 + m \cdot y_1 \bmod q\\ \sigma_2 &:= x_2 + m \cdot y_2 \bmod q\end{aligned}$$

Signature verification

$$p_1 p_2^m \stackrel{?}{=} g^{\sigma_1} h^{\sigma_2}$$

(in fact *m* should be a hash of the message to be signed)

The Failstop Signatures Based on DLP

extension for multiple-signatures

an extension from the same EUROCRYPT '92 paper

a secret key:

$$SK = (x_1, y_1, x_2, y_2, \ldots, x_{k+1}, y_{k+1})$$

the corresponding public key:

$$PK = (p_1, p_2, \ldots, p_{k+1}) = (g^{x_1} h^{y_1}, g^{x_2} h^{y_2}, \ldots, g^{x_{k+1}} h^{y_{k+1}})$$

The Failstop Signatures Based on DLP

extension for multiple-signatures

assume that $\mathcal{D}$ has signed $i-1$ messages ($1 \le i \le k$)

The *i*th signature created for message *m*:

$$\mathrm{Sign}(SK, m, i) = (i, \sigma_{1,i}, \sigma_{2,i})$$

where

$$\begin{aligned}\sigma_{1,i} &:= x_i + m \cdot x_{i+1}\\ \sigma_{2,i} &:= y_i + m \cdot y_{i+1}\end{aligned}$$

Verification:

$$p_i p_{i+1}^m \stackrel{?}{=} g^{\sigma_{1,i}} h^{\sigma_{2,i}}$$

The Failstop Signatures Based on DLP
basic properties

Lemma

Given k different signatures $(i, \sigma_{1,i}, \sigma_{2,i})$*, for* $i = 1, 2, \ldots, k$*, there are q possible secret keys that match* $p_1, p_2, \ldots, p_{k+1}$.

Lemma

Given signatures S_i *on* m *and* S_i' *on* $m' \neq m \bmod q$

Then there are unique (x_i, y_i), (x_{i+1}, y_{i+1})*, such that*

- $p_i = g^{x_i} h^{y_i}$, $p_{i+1} = g^{x_{i+1}} h^{y_{i+1}}$
- *both* S *and* S' *are created with* (x_i, y_i) *and* (x_{i+1}, y_{i+1})

The Failstop Signatures Based on DLP

basic properties

Lemma

If the presumed signer $\mathcal{D}$

- *receives a valid, forged signature* S_i' *on* m
- *creates a signature* S_i *on* m *with* $\mathcal{D}$*'s secret keys corresponding to* (p_i, p_{i+1})
- $S_i' \neq S_i$

then $\mathcal{D}$ *can compute* $\log_g(h)$.

Stamp&Extend

Ł. Krzywiecki, P. Kubiak, and M. Kutyłowski,
"Stamp and extend - instant but undeniable timestamping based on lazy trees" INTRUST 2012

an append-only archive based on Schnorr Signatures

- basic property: an ephemeral can be used only once, otherwise signing key leaked
- ephemerals committed in advance

Schnorr Signatures

David's keys

- private key is a random number x
- the corresponding public key is $y = g^x$

signing M

1. k chosen uniformly at random
2. $r := g^k$
3. $e := \mathrm{Hash}(r||M)$
4. $s := k - x \cdot e \bmod q$, where q is the group order
5. output signature (s, e) (notation $\mathrm{SDSA}_{x,k}(M)$)

then:

- k called an *ephemeral private key*
- r called an *ephemeral public key*

Schnorr Signatures

crucial property

Property

If signatures σ_1 and σ_2 have been created with ephemeral private keys k_1, k_2 such that $\delta = k_1 - k_2$ ***is known***

then

one can derive the private signing key from σ_1, σ_2 ***and*** δ

Customized Commitment Scheme

commitments for transaction *i*

user $\mathcal{D}$ creates two commitments

$$c_{2i}, c_{2i+1},$$

analogously to Stamp&Extend scheme. However, unlike before the public key y is used:

$$c_j = g^{k_j} y^{\ell_j}$$

for $j = 2i, 2i+1$.

$\ell_j = \mathrm{Hash}(c_j || M)$, where M is a message determined by the jth transaction request

So $k_j = \log_g(c_j) - x \cdot \ell_j \bmod q$ and opening the commitment c_j by revealing (ℓ_j, k_j) amounts to publishing the Schnorr signature on M

Commitment Security Mechanism

If x and $\log_g(c_j)$ are used to sign messages $M \neq M'$, then two pairs (ℓ_j, k_j), (ℓ'_j, k'_j) are revealed for the same c_j:

- if $\ell_j \neq \ell'_j$ mod q then, as $\log_g(c_j)$ is fixed, the private key x leaks via the formula:

$$x = (k_j - k'_j) \cdot (\ell'_j - \ell_j)^{-1} \bmod q,$$

- if $\ell_j = \ell'_j$ mod q, then we get a hash collision

$$\mathrm{Hash}(c_j||M) = \mathrm{Hash}(c_j||M') \bmod q$$

in both cases we get an evidence of a security breach

User Initialization

To get a certificate user $\mathcal{D}$ must:

1. generate a private key $x \in \mathbb{Z}_q \setminus \{0, 1\}$ and the public key $y = g^x$,
2. generate an ephemeral private key w_1 for signing the first transaction request, together with the corresponding commitment $c_1 = g^{w_1}$,
3. generate the initial keys for failstop signatures:

$$p_1 = g^{x_1} h^{y_1}, \quad p_2 = g^{x_2} h^{y_2}$$

where (x_1, y_1), and (x_2, y_2) are chosen uniformly at random.

The certificate $Cert_{\mathcal{D}}$ of $\mathcal{D}$ will contain y, c_1, p_1, p_2.

Creating a Transaction Request

Outline

- **sign the transaction using a Schnorr signature with a commitment already fixed** long ago
  (it prevents forking)
- **sign the transaction with failstop signature** with ephemeral keys fixed during the previous transaction,
  commit to the failstop key for the next transaction
  (it prevents later transaction manipulations)

Creating a Transaction Request

ith transaction request R_i of user $\mathcal{D}$:

a pair:

$$R_i = (T_i, D_i)$$

where

- D_i is the failstop signature on T_i created with the secret keys corresponding to p_i, p_{i+1}, respectively.
- $T_i = (T_i', \mathrm{SDSA}_{x,w_i}(T_i'))$
  - where the ephemeral private key w_i has been committed as $c_i = g^{w_i}$ in the request $R_{\lfloor i/2 \rfloor}$.
- $T_i' = (c_{2i}, c_{2i+1}, i, \mathrm{Hash}(R_{i-1}), t_i, p_{i+2})$

Creating a Transaction Request

cnt

T_i' is defined as

$$T_i' = (c_{2i}, c_{2i+1}, i, \mathrm{Hash}(R_{i-1}), t_i, p_{i+2}).$$

In more detail:

- c_{2i}, c_{2i+1} are commitments to ephemeral private keys w_{2i}, w_{2i+1} to be used in the future (to sign T_{2i}', T_{2i+1}'),
- i is the request number
- $\mathrm{Hash}(R_{i-1})$ is the hash of the previous transaction request of $\mathcal{D}$,
- t_i is the transaction data,
- $p_{i+2} = g^{x_{i+2}} h^{y_{i+2}}$ is a new failstop signature public key for a secret key (x_{i+2}, y_{i+2}) chosen uniformly at random.

The Situation of Goliath

Goliath can break DLP, so

- he can derive the secret key x from the public key of a user $\mathcal{D}$,
- he can derive the ephemeral private key w_j for any commitment $c_j = g^{w_j}$ created by $\mathcal{D}$,
- he can compute $a = \log_g(h)$.

can Goliath impersonate $\mathcal{D}$ and add a transaction on behalf of $\mathcal{D}$?

Goliath impersonating David

manipulating a transaction

Assume that $\mathcal{D}$ has made $i-1$ transactions.
Goliath is attempting to create a valid request $\widetilde{R}_j = (\widetilde{T}_j, \widetilde{D}_j)$ for a $j \le i$:

- he can prepare any $\widetilde{T}_j'$ of the required form,
- knowing x and w_j he can create $\widetilde{T}_j = (\widetilde{T}_j', \mathrm{SDSA}_{x,w_j}(\widetilde{T}_j'))$,
- he can prepare a failstop signature $\widetilde{D}_j$ on $\widetilde{T}_j$. However:
  - for keys $p_1, p_2, \ldots, p_{i+1}$ and failstop signatures $D_1, D_2, \ldots, D_{i-1}$ already created by $\mathcal{D}$, Goliath **cannot determine $SK_j = (x_j, y_j)$, $SK_{j+1} = (x_{j+1}, y_{j+1})$ used by $\mathcal{D}$**
  - ... so Goliath **must choose his version of the secret keys**, say $\widetilde{SK}_j = (\widetilde{x}_j, \widetilde{y}_j)$, $\widetilde{SK}_{j+1} = (\widetilde{x_{j+1}}, \widetilde{y_{j+1}})$:
    - for $\widetilde{x}_j$ chosen at random, Goliath computes $\widetilde{y}_j$ from $p_j = g^{\widetilde{x}_j} h^{\widetilde{y}_j}$ (Goliath can compute discrete logs!)
    - . . .

Goliath impersonating David

manipulating a transaction

- ... so Goliath must choose his version of the secret keys, say $\widetilde{SK}_j = (\tilde{x}_j, \tilde{y}_j)$, $\widetilde{SK}_{j+1} = (\widetilde{x_{j+1}}, \widetilde{y_{j+1}})$:
  - for $\tilde{x}_j$ chosen at random, Goliath computes $\tilde{y}_j$ from $p_j = g^{\tilde{x}_j} h^{\tilde{y}_j}$ (Goliath can compute discrete logs!)
  - if $j < i$, then on the basis of signature D_j he can determine $\widetilde{x_{j+1}}$ and $\widetilde{y_{j+1}}$,
  - if $j = i$, Goliath chooses $\widetilde{x_{j+1}}$ at random and computes $\widetilde{y_{j+1}}$ from $p_{j+1} = g^{\widetilde{x_{j+1}}} h^{\widetilde{y_{j+1}}}$

Alternatively, for an existing T_j Goliath may try to generate $\widetilde{T}_j$ such that $\mathrm{Hash}(T_j) = \mathrm{Hash}(\widetilde{T}_j)$ mod q and skip generating own keys $\widetilde{SK}_j$ and $\widetilde{SK}_{j+1}$.

David proving manipulation

$\mathcal{D}$ proves that the transaction $\widetilde{R}_j$ has been forged:

If $j < i$, then $\mathcal{D}$ takes his original T_j and at first checks if $\mathrm{Hash}(T_j) = \mathrm{Hash}(\widetilde{T}_j)$ mod q:

- If yes, then there is conflict of Hash that must have been computed by Goliath,
- otherwise: there are signatures D_j and $\widetilde{D}_j$ corresponding to (p_j, p_{j+1}) but for different messages $m = \mathrm{Hash}(T_j)$ and $\widetilde{m} = \mathrm{Hash}(\widetilde{T}_j)$
- By Lemma, these signatures uniquely determine the secret keys SK_j^*, SK_{j+1}^* corresponding to $(D_j, \widetilde{D}_j, m, \widetilde{m})$.

David proving manipulation

cnt

- Note that for given $\widetilde{m}$ the choice of $\widetilde{SK}_j$, $\widetilde{SK}_{j+1}$ uniquely determines $\widetilde{D}_j$, so also uniquely determines SK^*_j, SK^*_{j+1} given (D_j, m).
- But Goliath had at least q degrees of freedom in choosing $\widetilde{SK}_j$, $\widetilde{SK}_{j+1}$, so it is unlikely that $(SK^*_j, SK^*_{j+1}) = (SK_j, SK_{j+1})$.
- Now it suffices that $\mathcal{D}$ generates a signature $\hat{D}_j$ on $\widetilde{m}$ using his own keys SK_j, SK_{j+1} and according to Lemma $\mathcal{D}$ finds $\log_g h$

Case $j = i$

$\mathcal{D}$ generates a new T_j and finds $\log_g h$ in the same way.

Situation of Goliath

forking a chain

what happens if the list has been forked?

- there is a position i where forking occurs
- there are two Schnorr signatures corresponding to this position
- both signatures are based on the same commitment and therefore the same component $r = g^k$ of the Schnorr signature
- $\Rightarrow$ the signing key x can be derived

Conclusion

- **an upgrade for the simple Dolev-Liber blockchain**
- **one should not fear of an all-mighty adversary:**

any forgery attempt will be discovered and proved

so

the adversary's cryptanalytic advantage turns out to be useless

Thanks for your attention!