# Timed Signatures and Zero-Knowledge Proofs –Timestamping in the Blockchain Era–


Aydin Abadi

Aggelos Kiayias

Michele Ciampi

Vassilis Zikas

# Timestamping

- The time stamping process provides a temporal order among a set of events
- The newspapers have this feature

# Why is Timestamping important?

- Proving the date of content creation (e.g. patents, keeping track of the history of goods)
- Markets in Financial Instruments Directive II (MiFID II): Any event required for trading venues has to be timestamped accordingly to a unique clock.

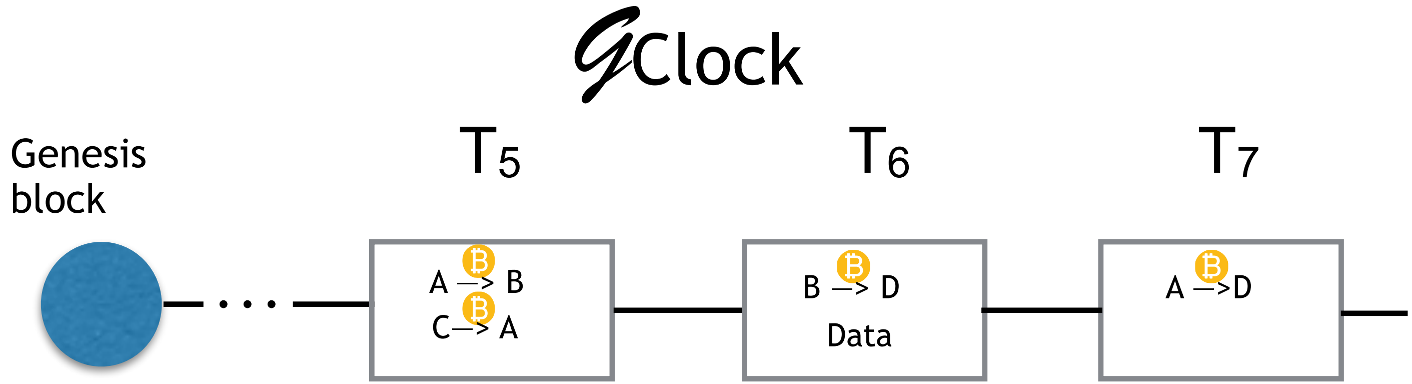
# State of the art

- Server aided model [Haber, Stornetta ’91]
- The previous approach has been improved in many aspects (e.g. [Buldas, Laanoja, Truu '17])
- Distributed scenarios:
  1. the documents can be jointly signed by n parties (incentive issues)
  2. using the blockchain (e.g. [Clark, Essex ’12] OriginStamp, Guardtime)
- UC formalizations:
  - Server aided model [MSTS04]
  - Non-interactive timestamping via VDF [LSS19]

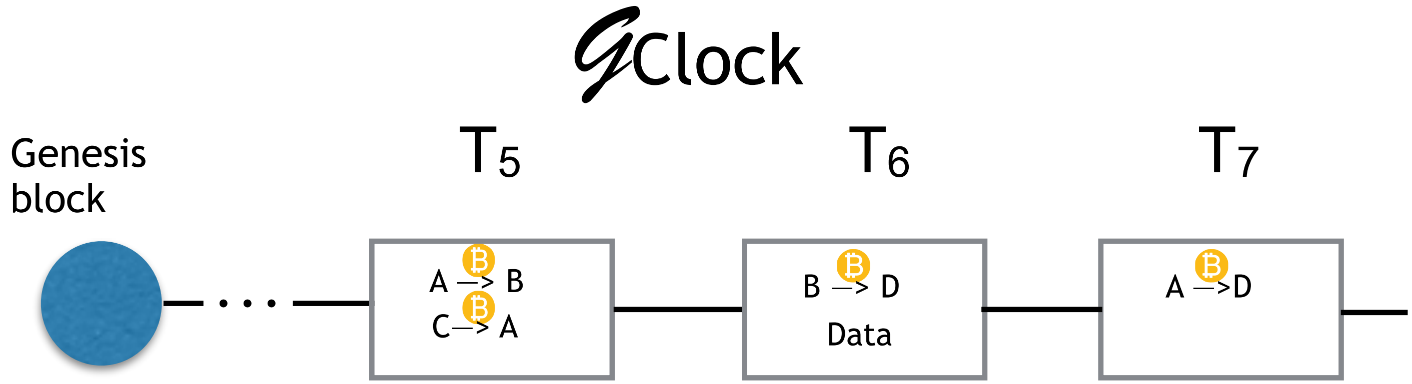
Tal Moran, Ronen Shaltiel, Amnon Ta-Shma: JoC

Landerreche, Schaffner, Stevens: ePrint

# Timestamping via a distributed ledger

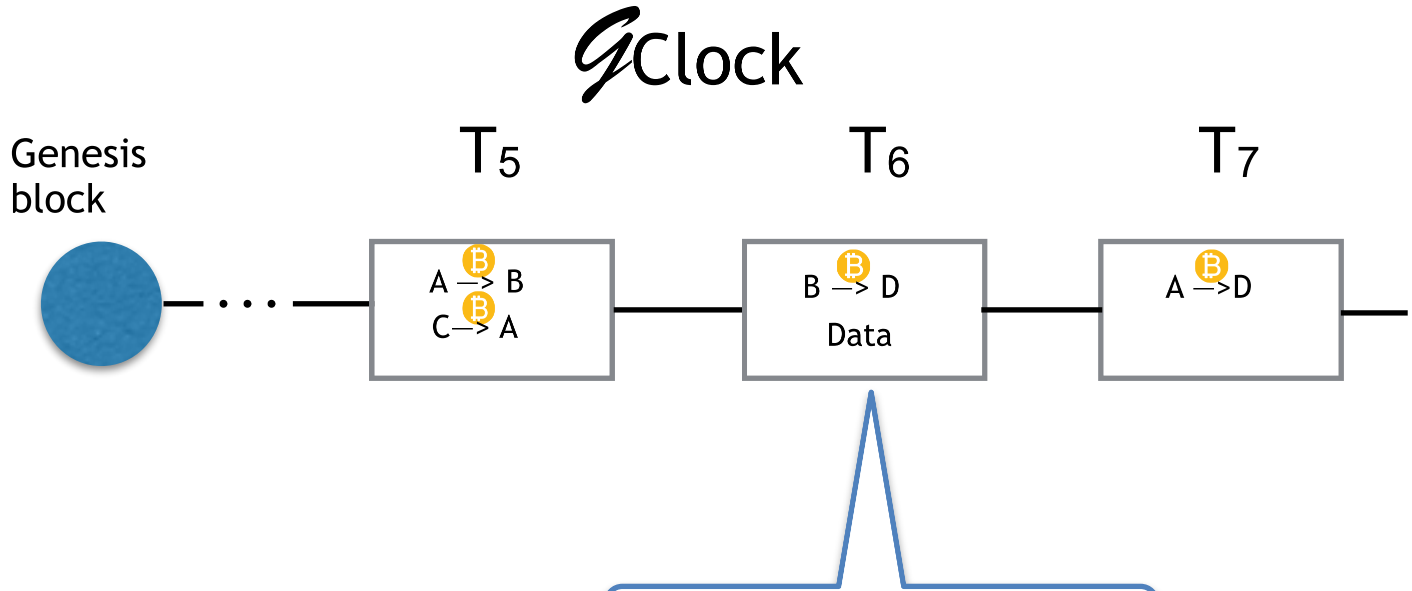


# Timestamping via a distributed ledger




- Liveness
- Immutability

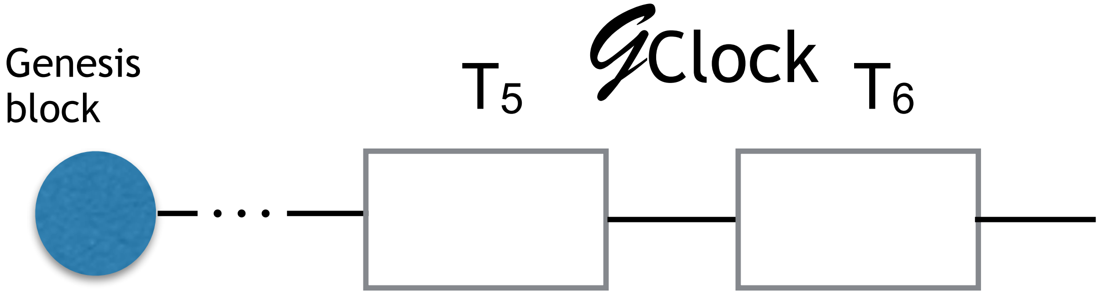
# Timestamping via a distributed ledger

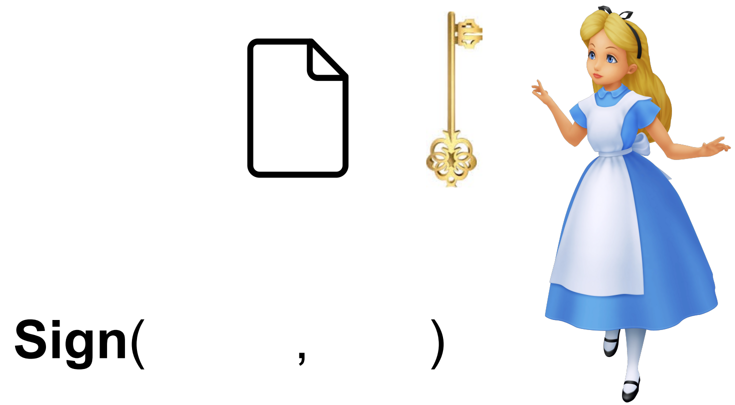



Ledger functionality proposed in [BMTZ CRYPTO 17]

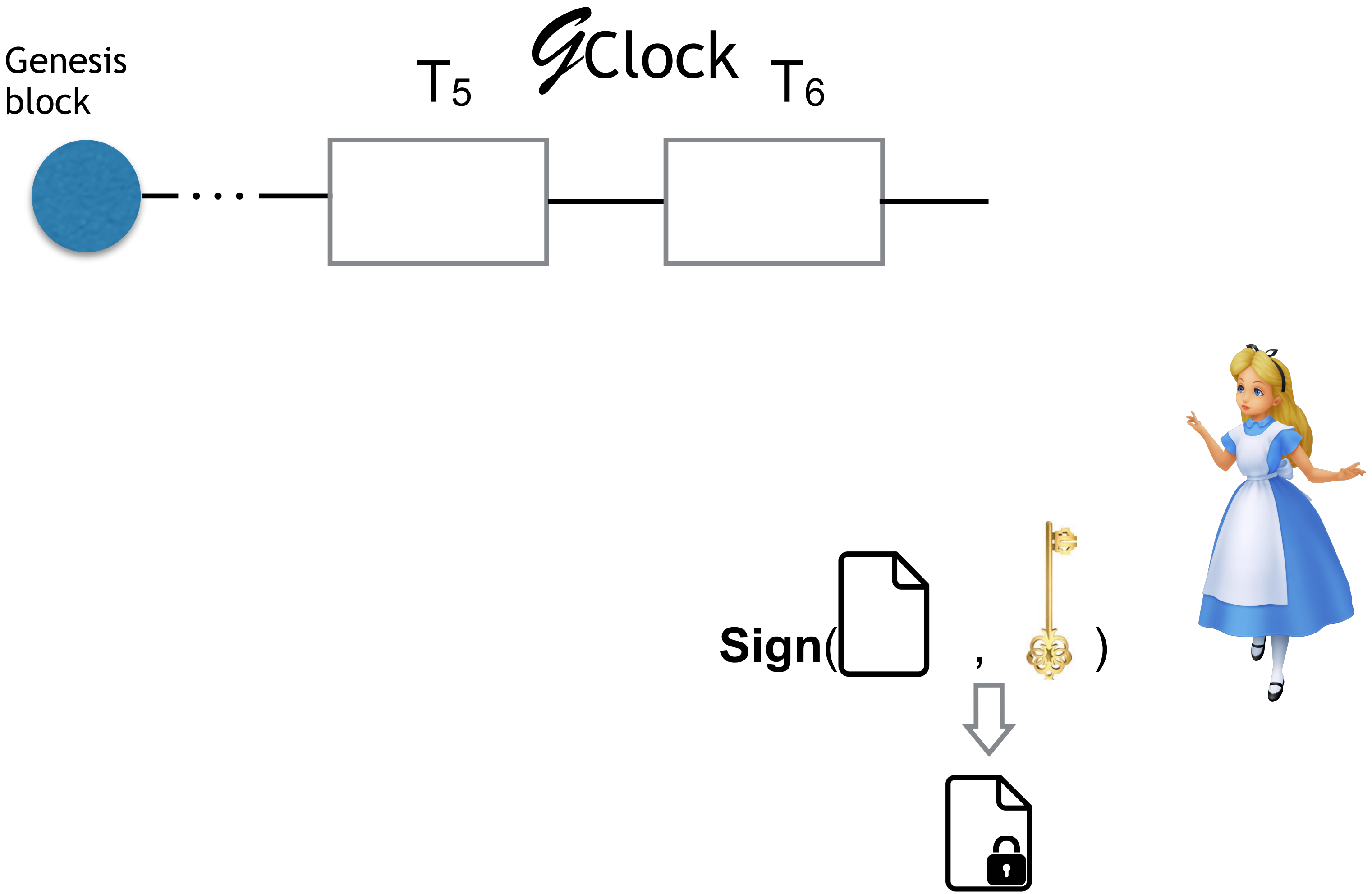
- Liveness
- Immutability

# Backdate security via a distributed ledger

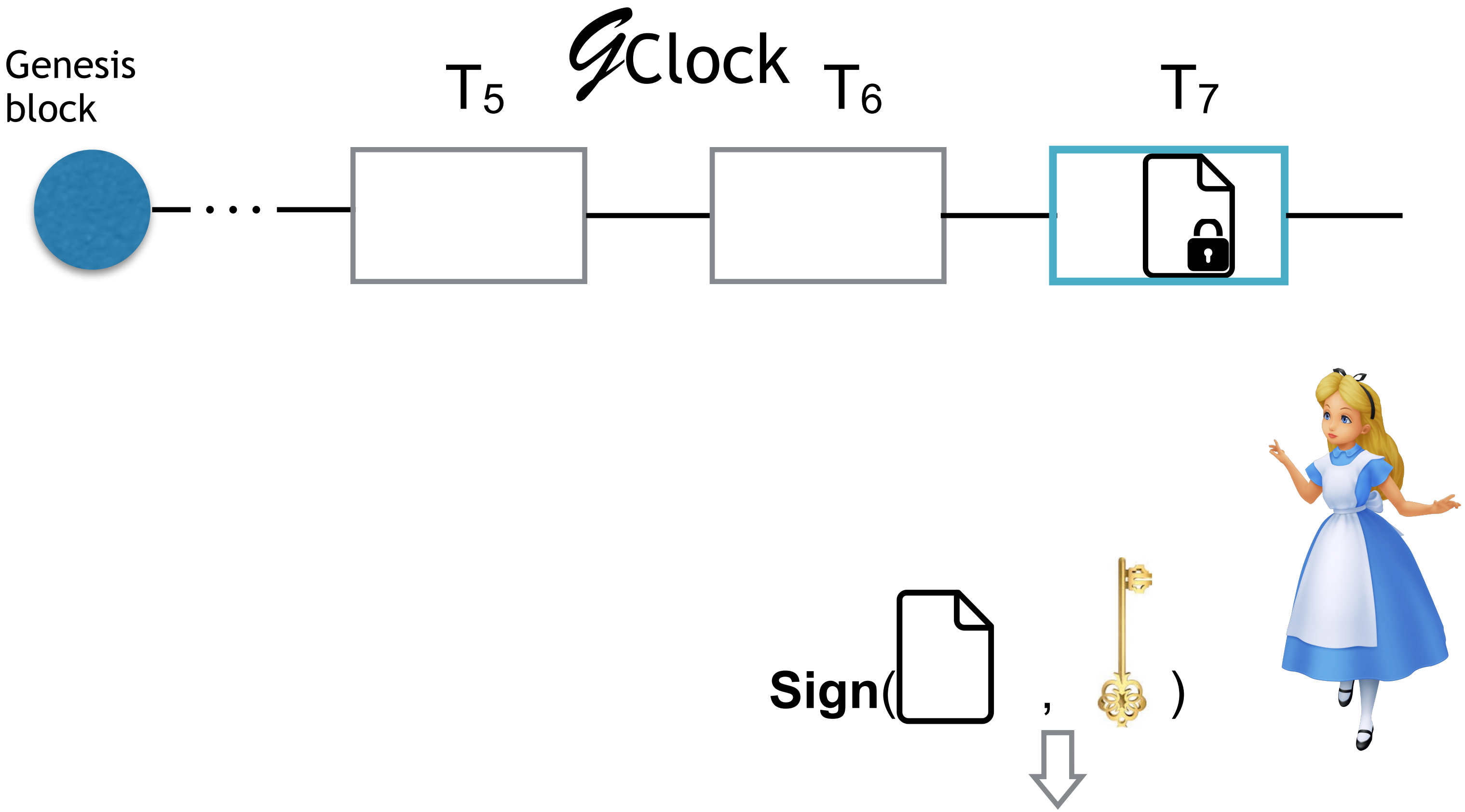


# Backdate security via a distributed ledger



# Backdate security via a distributed ledger

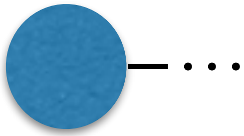


# Backdate security via a distributed ledger

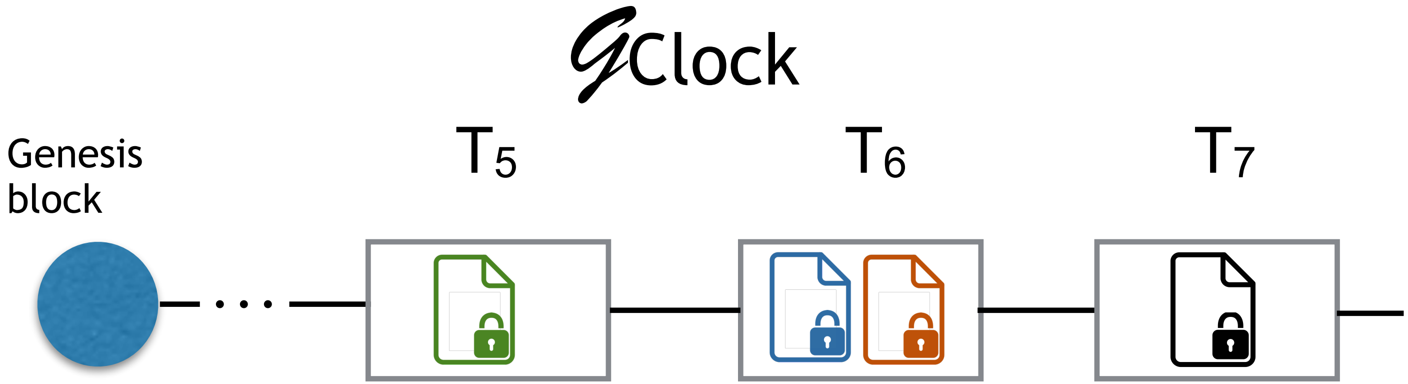
$\mathcal{G}$Clock

Genesis block

$T_5$ $T_6$ $T_7$

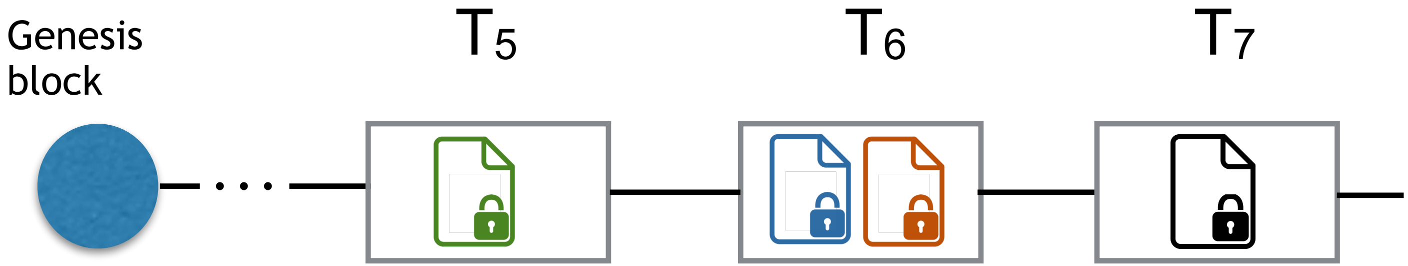


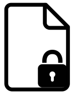
# Backdate security via a distributed ledger



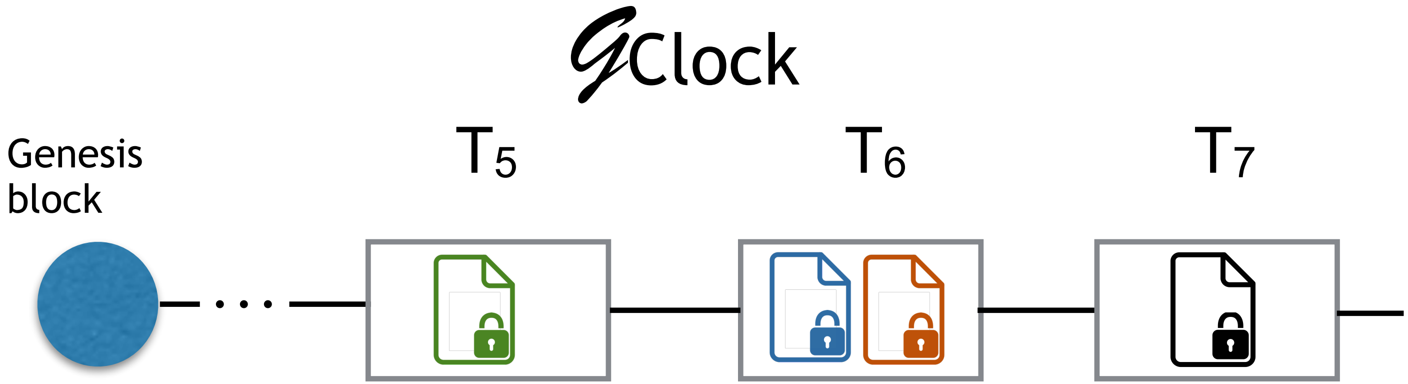

# Backdate security via a distributed ledger

$\mathcal{G}$Clock

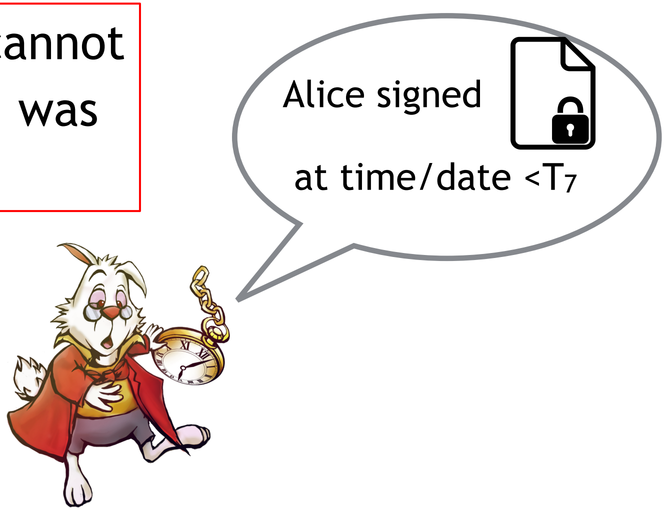



At time $T_7$ the rabbit cannot be convinced that  was signed before 

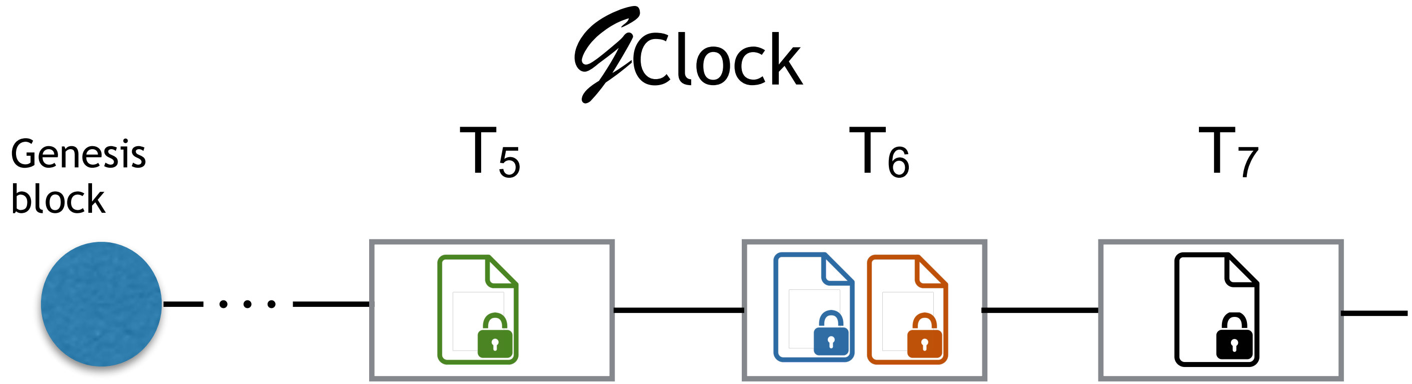
# Backdate security via a distributed ledger



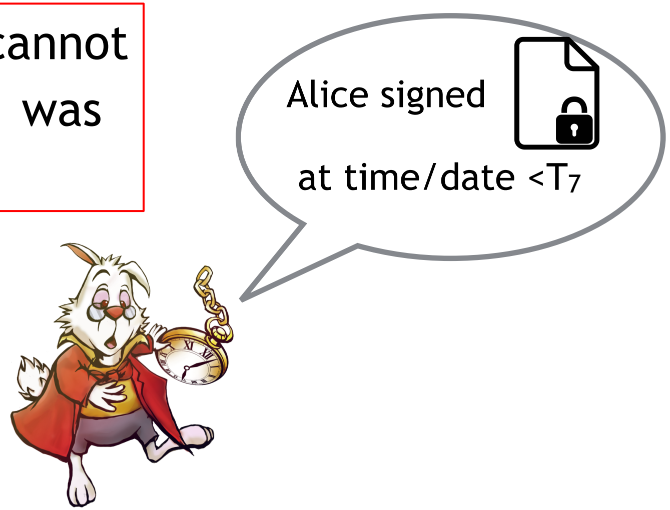

At time $T_7$ the rabbit cannot be convinced that [black document] was signed before [orange document]



# Backdate security via a distributed ledger

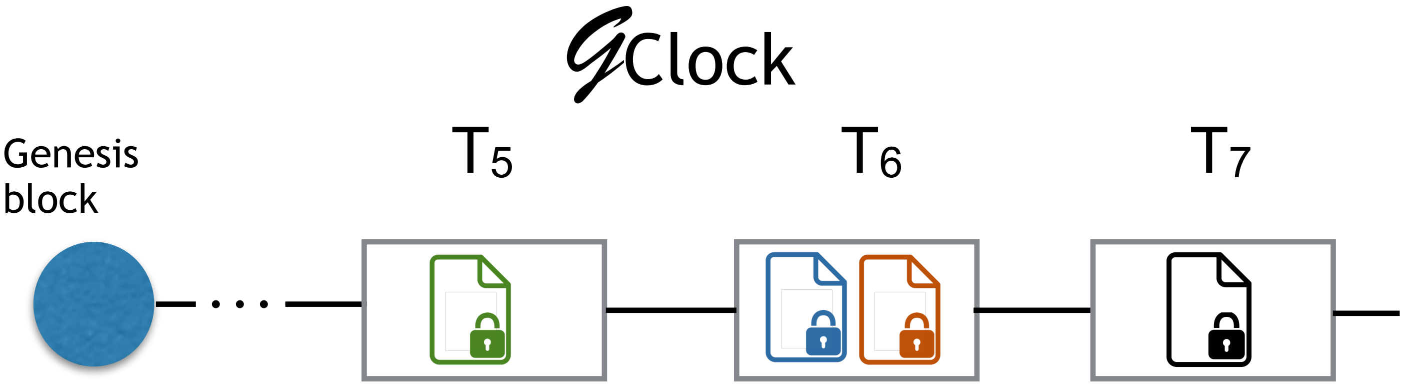



At time $T_7$ the rabbit cannot be convinced that [document] was signed before [document]

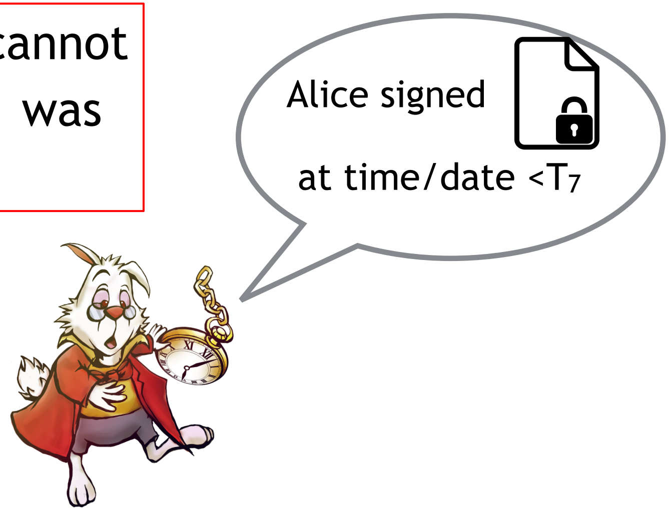



This protocol allows to postdate a signature

# Backdate security via a distributed ledger




At time $T_7$ the rabbit cannot be convinced that was signed before




This protocol allows to postdate a signature

Let's embed un unpredictable random value inside the signature

# The Ledger has no source of randomness

**The ledger functionality is aimed to abstract the consensus layer of Blockchains!**

# The Ledger has no source of randomness

**The ledger functionality is aimed to abstract the consensus layer of Blockchains!**

**But Bitcoin/Ouroboros/Algorand… protocol does have "fresh" randomness!**

- The *honest* miners' nonces
  - Implicit in all security proofs
- The *honest miners' keys*
- *The honest miners' transactions*

# The Ledger has no source of randomness

**The ledger functionality is aimed to abstract the consensus layer of Blockchains!**

**But Bitcoin/Ouroboros/Algorand… protocol does have "fresh" randomness!**

- The *honest* miners' nonces
    - Implicit in all security proofs
- The *honest miners' keys*
- *The honest miners' transactions*

**Since Bitcoin/Ouroboros implements the ledger, its contents would have similar properties**

# How can we add randomness to ledger?

Let’s design a new ledger

# How can we add randomness to ledger?

## Let’s design a new ledger

**Functionality $\mathcal{G}_{\text{LEDGER}}$**

**General:** The functionality is parametrized by four algorithms Validate, ExtendPolicy, Blockify, and predict-time, along with two parameters windowSize, Delay $\in \mathbb{N}$. The functionality manages variables state, NxtBC, buffer, $\tau_L$, and $\vec{\tau}_{\text{state}}$, as described above. Initially, state := $\vec{\tau}_{\text{state}}$ := NxtBC := $\varepsilon$, buffer := $\emptyset$, $\tau_L = 0$.

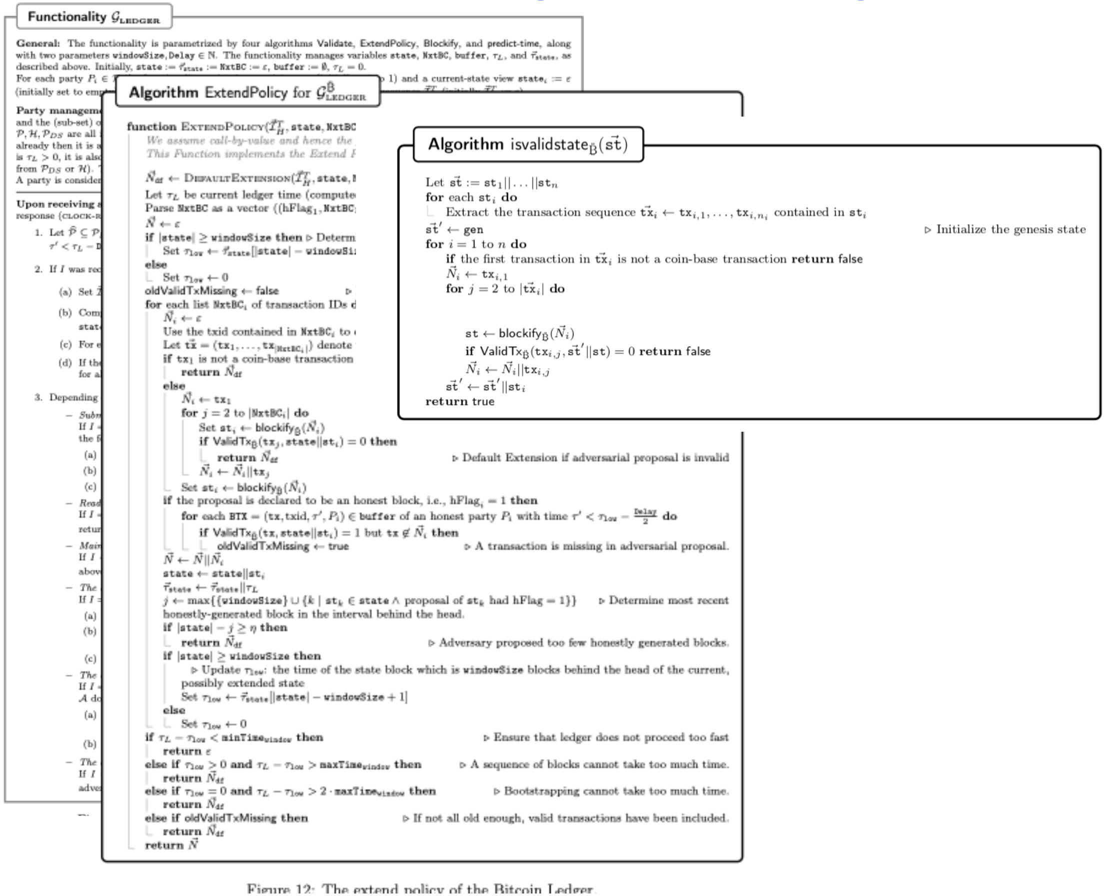
**Algorithm** ExtendPolicy for $\mathcal{G}^{\mathcal{B}}_{\text{LEDGER}}$

```
function ExtendPolicy(I_H^T, state, NxtBC, buffer, τ_state)
  We assume call-by-value and hence the function has no side effects.
  This Function implements the Extend Policy of the Bitcoin Ledger.
  N_df ← DefaultExtension(I_H^T, state, NxtBC, buffer, τ_state)   ▷ Extension if adversary violates policy.
  Let τ_L be current ledger time (computed from I_H^T)
  Parse NxtBC as a vector ((hFlag_1, NxtBC_1), ..., (hFlag_n, NxtBC_n))
  N ← ε                                                              ▷ Initialize Result
  if |state| ≥ windowSize then ▷ Determine time of the block which is windowSize blocks behind the state head
    Set τ_low ← τ_state[|state| − windowSize + 1]
  else
    Set τ_low ← 0
  oldValidTxMissing ← false        ▷ Flag to keep track whether old enough, valid transactions are inserted.
  for each list NxtBC_i of transaction IDs do              ▷ Compute the next state block and verify validity
    N_i ← ε
    Use the txid contained in NxtBC_i to determine the list of transactions
    Let tx = (tx_1, ..., tx_|NxtBC_i|) denote the transactions of NxtBC_i
    if tx_1 is not a coin-base transaction then
      return N_df
    else
      N_i ← tx_1
      for j = 2 to |NxtBC_i| do
        Set st_i ← blockify_B(N_i)
        if ValidTx_B(tx_j, state||st_i) = 0 then
          return N_df                        ▷ Default Extension if adversarial proposal is invalid
        N_i ← N_i||tx_j
      Set st_i ← blockify_B(N_i)
    if the proposal is declared to be an honest block, i.e., hFlag_i = 1 then
      for each BTX = (tx, txid, τ', P_i) ∈ buffer of an honest party P_i with time τ' < τ_low − Delay/2 do
        if ValidTx_B(tx, state||st_i) = 1 but tx ∉ N_i then
          oldValidTxMissing ← true           ▷ A transaction is missing in adversarial proposal.
    N ← N||N_i
    state ← state||st_i
    τ_state ← τ_state||τ_L
    j ← max({windowSize} ∪ {k | st_k ∈ state ∧ proposal of st_k had hFlag = 1})   ▷ Determine most recent
    honestly-generated block in the interval behind the head.
    if |state| − j ≥ η then
      return N_df                            ▷ Adversary proposed too few honestly generated blocks.
    if |state| ≥ windowSize then
      ▷ Update τ_low: the time of the state block which is windowSize blocks behind the head of the current,
      possibly extended state
      Set τ_low ← τ_state[|state| − windowSize + 1]
    else
      Set τ_low ← 0
  if τ_L − τ_low < minTime_window then                  ▷ Ensure that ledger does not proceed too fast
    return ε
  else if τ_low > 0 and τ_L − τ_low > maxTime_window then   ▷ A sequence of blocks cannot take too much time.
    return N_df
  else if τ_low = 0 and τ_L − τ_low > 2 · maxTime_window then   ▷ Bootstrapping cannot take too much time.
    return N_df
  else if oldValidTxMissing then     ▷ If not all old enough, valid transactions have been included.
    return N_df
  return N
```

Figure 12: The extend policy of the Bitcoin Ledger.

# How can we add randomness to ledger?

## Let's design a new ledger



Functionality $\mathcal{G}_{\text{LEDGER}}$

**General:** The functionality is parametrized by four algorithms Validate, ExtendPolicy, Blockify, and predict-time, along with two parameters windowSize, Delay $\in \mathbb{N}$. The functionality manages variables state, NxtBC, buffer, $\tau_L$, and $\vec{\tau}_{\text{state}}$, as described above. Initially, state := $\vec{\tau}_{\text{state}}$ := NxtBC := $\varepsilon$, buffer := $\emptyset$, $\tau_L = 0$.

Algorithm ExtendPolicy for $\mathcal{G}^{\mathcal{B}}_{\text{LEDGER}}$

```
function ExtendPolicy(I_H^T, state, NxtBC ...
  N_df ← DefaultExtension(I_H^T, state, ...
  Let τ_L be current ledger time (compute...
  Parse NxtBC as a vector ((hFlag_1, NxtBC_1 ...
  N ← ε
  if |state| ≥ windowSize then ▷ Determ...
    Set τ_low ← τ_state[|state| − windowSi...
  else
    Set τ_low ← 0
  oldValidTxMissing ← false
  for each list NxtBC_i of transaction IDs d...
    N_i ← ε
    Use the txid contained in NxtBC_i to ...
    Let tx = (tx_1, ..., tx_|NxtBC_i|) denote
    if tx_1 is not a coin-base transaction
      return N_df
    else
      N_i ← tx_1
      for j = 2 to |NxtBC_i| do
        Set st_i ← blockify_B(N_i)
        if ValidTx_B(tx_j, state||st_i) = 0 then
          return N_df        ▷ Default Extension if adversarial proposal is invalid
        N_i ← N_i||tx_j
      Set st_i ← blockify_B(N_i)
    if the proposal is declared to be an honest block, i.e., hFlag_i = 1 then
      for each BTX = (tx, txid, τ', P_i) ∈ buffer of an honest party P_i with time τ' < τ_low − Delay/2 do
        if ValidTx_B(tx, state||st_i) = 1 but tx ∉ N_i then
          oldValidTxMissing ← true        ▷ A transaction is missing in adversarial proposal.
    N ← N||N_i
    state ← state||st_i
    τ_state ← τ_state||τ_L
    j ← max({windowSize} ∪ {k | st_k ∈ state ∧ proposal of st_k had hFlag = 1})   ▷ Determine most recent honestly-generated block in the interval behind the head.
    if |state| − j ≥ η then
      return N_df        ▷ Adversary proposed too few honestly generated blocks.
    if |state| ≥ windowSize then
      ▷ Update τ_low: the time of the state block which is windowSize blocks behind the head of the current, possibly extended state
      Set τ_low ← τ_state[|state| − windowSize + 1]
    else
      Set τ_low ← 0
  if τ_L − τ_low < minTime_window then        ▷ Ensure that ledger does not proceed too fast
    return ε
  else if τ_low > 0 and τ_L − τ_low > maxTime_window then        ▷ A sequence of blocks cannot take too much time.
    return N_df
  else if τ_low = 0 and τ_L − τ_low > 2 · maxTime_window then        ▷ Bootstrapping cannot take too much time.
    return N_df
  else if oldValidTxMissing then        ▷ If not all old enough, valid transactions have been included.
    return N_df
  return N
```

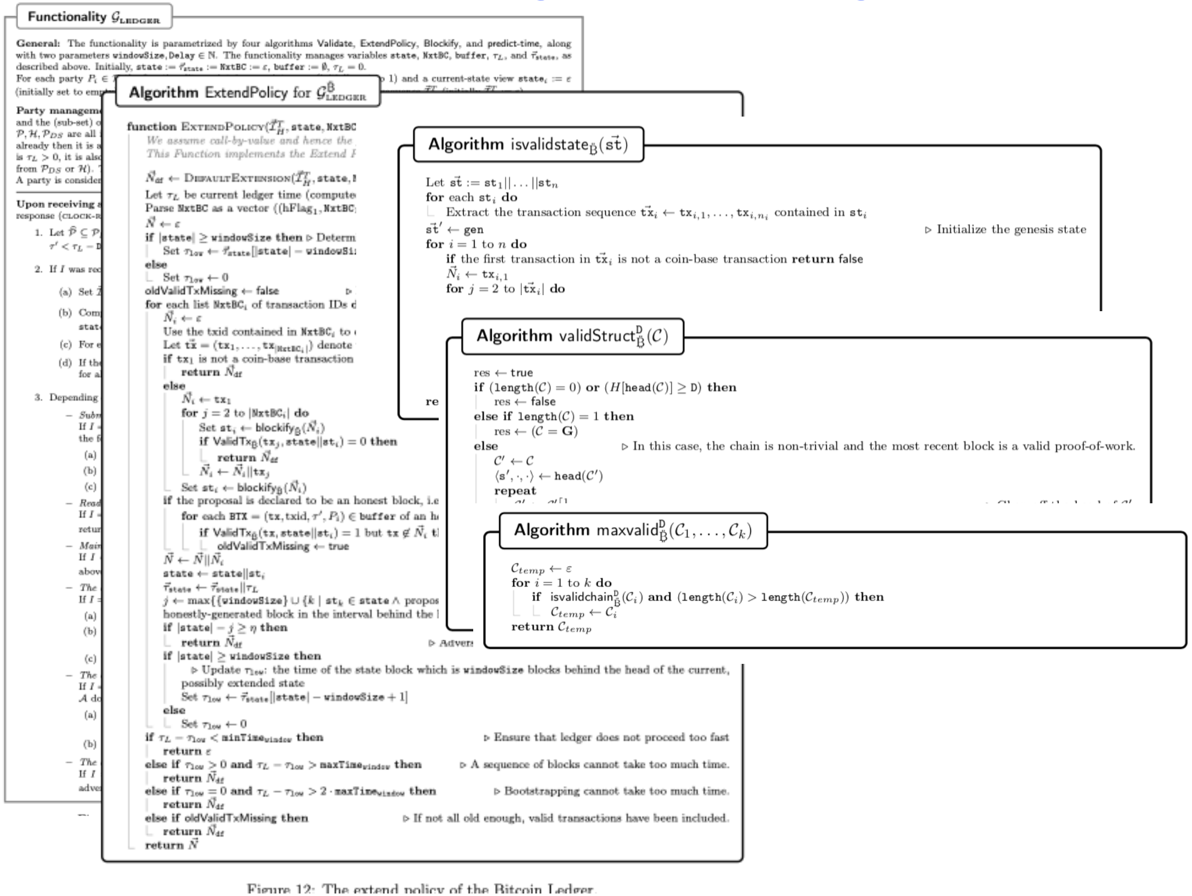
Algorithm isvalidstate$_{\mathcal{B}}(\vec{st})$

```
Let st := st_1||...||st_n
for each st_i do
  Extract the transaction sequence tx_i ← tx_{i,1}, ..., tx_{i,n_i} contained in st_i
st' ← gen        ▷ Initialize the genesis state
for i = 1 to n do
  if the first transaction in tx_i is not a coin-base transaction return false
  N_i ← tx_{i,1}
  for j = 2 to |tx_i| do
    st ← blockify_B(N_i)
    if ValidTx_B(tx_{i,j}, st'||st) = 0 return false
    N_i ← N_i||tx_{i,j}
  st' ← st'||st_i
return true
```

Figure 12: The extend policy of the Bitcoin Ledger.

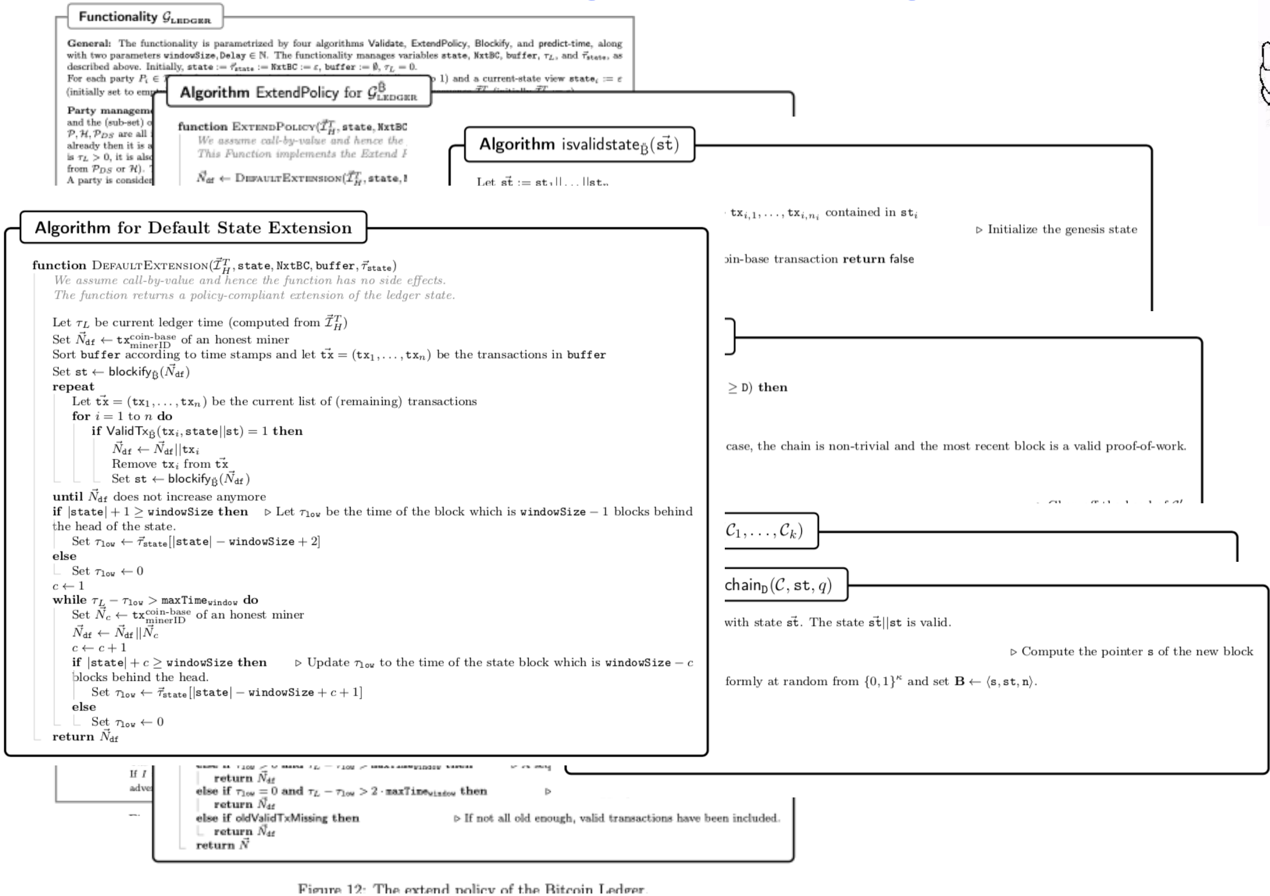
# How can we add randomness to ledger?

## Let's design a new ledger



# How can we add randomness to ledger?

## Let's design a new ledger



Functionality $\mathcal{G}_{\text{LEDGER}}$

Algorithm ExtendPolicy for $\mathcal{G}^{\mathbb{B}}_{\text{LEDGER}}$

Algorithm $\mathsf{isvalidstate}_{\mathbb{B}}(\vec{\mathtt{st}})$

**Algorithm for Default State Extension**

**function** DefaultExtension($\vec{\mathcal{I}}_H^T$, state, NxtBC, buffer, $\vec{\tau}_{\mathtt{state}}$)

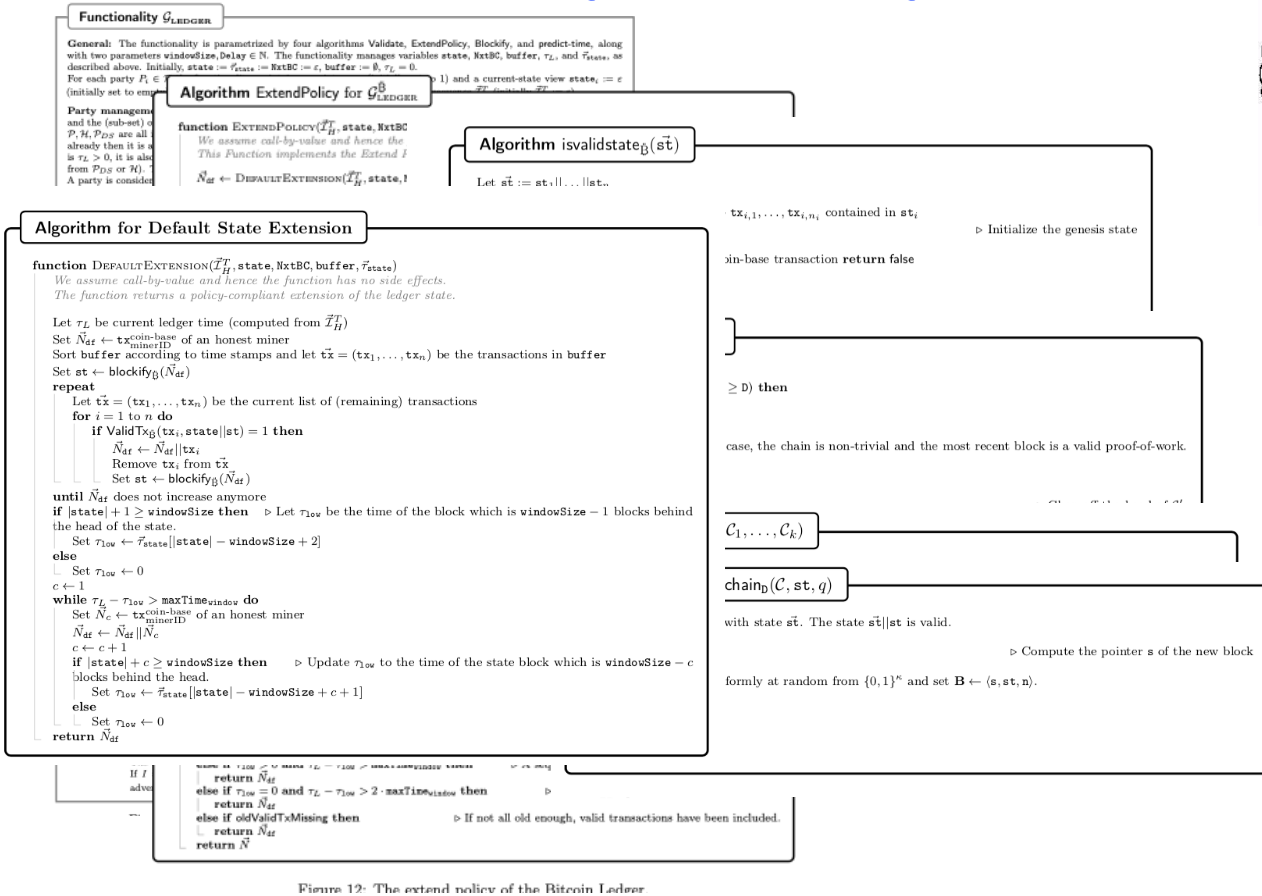
*We assume call-by-value and hence the function has no side effects.*
*The function returns a policy-compliant extension of the ledger state.*

Let $\tau_L$ be current ledger time (computed from $\vec{\mathcal{I}}_H^T$)
Set $\vec{N}_{\mathtt{df}} \leftarrow \mathtt{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
Sort buffer according to time stamps and let $\vec{\mathtt{tx}} = (\mathtt{tx}_1, \ldots, \mathtt{tx}_n)$ be the transactions in buffer
Set st $\leftarrow \mathsf{blockify}_{\mathbb{B}}(\vec{N}_{\mathtt{df}})$
**repeat**
  Let $\vec{\mathtt{tx}} = (\mathtt{tx}_1, \ldots, \mathtt{tx}_n)$ be the current list of (remaining) transactions
  **for** $i = 1$ to $n$ **do**
    **if** $\mathsf{ValidTx}_{\mathbb{B}}(\mathtt{tx}_i, \mathtt{state}||\mathtt{st}) = 1$ **then**
      $\vec{N}_{\mathtt{df}} \leftarrow \vec{N}_{\mathtt{df}}||\mathtt{tx}_i$
      Remove $\mathtt{tx}_i$ from $\vec{\mathtt{tx}}$
      Set st $\leftarrow \mathsf{blockify}_{\mathbb{B}}(\vec{N}_{\mathtt{df}})$
**until** $\vec{N}_{\mathtt{df}}$ does not increase anymore
**if** $|\mathtt{state}| + 1 \geq \mathtt{windowSize}$ **then** ▷ Let $\tau_{\text{low}}$ be the time of the block which is $\mathtt{windowSize} - 1$ blocks behind the head of the state.
  Set $\tau_{\text{low}} \leftarrow \vec{\tau}_{\mathtt{state}}[|\mathtt{state}| - \mathtt{windowSize} + 2]$
**else**
  Set $\tau_{\text{low}} \leftarrow 0$
$c \leftarrow 1$
**while** $\tau_L - \tau_{\text{low}} > \mathtt{maxTime}_{\mathtt{window}}$ **do**
  Set $\vec{N}_c \leftarrow \mathtt{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
  $\vec{N}_{\mathtt{df}} \leftarrow \vec{N}_{\mathtt{df}}||\vec{N}_c$
  $c \leftarrow c + 1$
  **if** $|\mathtt{state}| + c \geq \mathtt{windowSize}$ **then** ▷ Update $\tau_{\text{low}}$ to the time of the state block which is $\mathtt{windowSize} - c$ blocks behind the head.
    Set $\tau_{\text{low}} \leftarrow \vec{\tau}_{\mathtt{state}}[|\mathtt{state}| - \mathtt{windowSize} + c + 1]$
  **else**
    Set $\tau_{\text{low}} \leftarrow 0$
**return** $\vec{N}_{\mathtt{df}}$

Figure 12: The extend policy of the Bitcoin Ledger.

# How can we add randomness to ledger?

## Let’s design a new ledger



**Algorithm for Default State Extension**

**function** DefaultExtension($\vec{\mathcal{I}}_H^T$, state, NxtBC, buffer, $\vec{\tau}_{\text{state}}$)
*We assume call-by-value and hence the function has no side effects.*
*The function returns a policy-compliant extension of the ledger state.*
Let $\tau_L$ be current ledger time (computed from $\vec{\mathcal{I}}_H^T$)
Set $\vec{N}_{\text{df}} \leftarrow \text{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
Sort buffer according to time stamps and let $\vec{\text{tx}} = (\text{tx}_1, \dots, \text{tx}_n)$ be the transactions in buffer
Set st $\leftarrow$ blockify$_{\mathcal{B}}(\vec{N}_{\text{df}})$
**repeat**
Let $\vec{\text{tx}} = (\text{tx}_1, \dots, \text{tx}_n)$ be the current list of (remaining) transactions
**for** $i = 1$ to $n$ **do**
**if** ValidTx$_{\mathcal{B}}$(tx$_i$, state||st) = 1 **then**
$\vec{N}_{\text{df}} \leftarrow \vec{N}_{\text{df}} || \text{tx}_i$
Remove tx$_i$ from $\vec{\text{tx}}$
Set st $\leftarrow$ blockify$_{\mathcal{B}}(\vec{N}_{\text{df}})$
**until** $\vec{N}_{\text{df}}$ does not increase anymore
**if** |state| + 1 $\geq$ windowSize **then** ▷ Let $\tau_{\text{low}}$ be the time of the block which is windowSize − 1 blocks behind the head of the state.
Set $\tau_{\text{low}} \leftarrow \vec{\tau}_{\text{state}}$[|state| − windowSize + 2]
**else**
Set $\tau_{\text{low}} \leftarrow 0$
$c \leftarrow 1$
**while** $\tau_L - \tau_{\text{low}} >$ maxTime$_{\text{window}}$ **do**
Set $\vec{N}_c \leftarrow \text{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
$\vec{N}_{\text{df}} \leftarrow \vec{N}_{\text{df}} || \vec{N}_c$
$c \leftarrow c + 1$
**if** |state| + $c \geq$ windowSize **then** ▷ Update $\tau_{\text{low}}$ to the time of the state block which is windowSize − $c$ blocks behind the head.
Set $\tau_{\text{low}} \leftarrow \vec{\tau}_{\text{state}}$[|state| − windowSize + $c$ + 1]
**else**
Set $\tau_{\text{low}} \leftarrow 0$
**return** $\vec{N}_{\text{df}}$



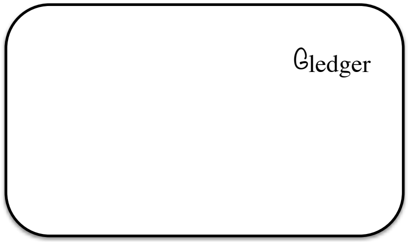
# How can we add randomness to ledger?

## Let’s design a new ledger


**And re-prove security from scratch …**

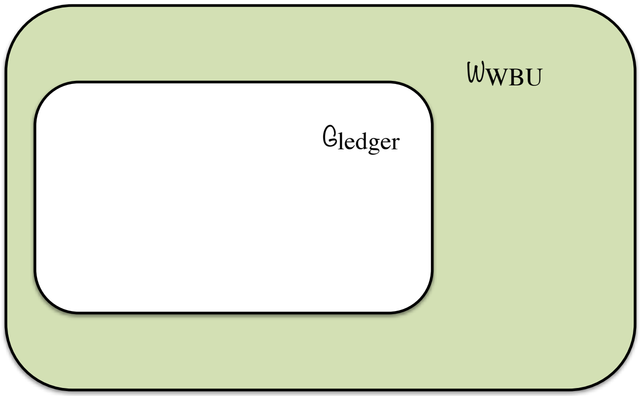
# How can we add randomness to ledger?

## Adding Randomness without changing the ledger



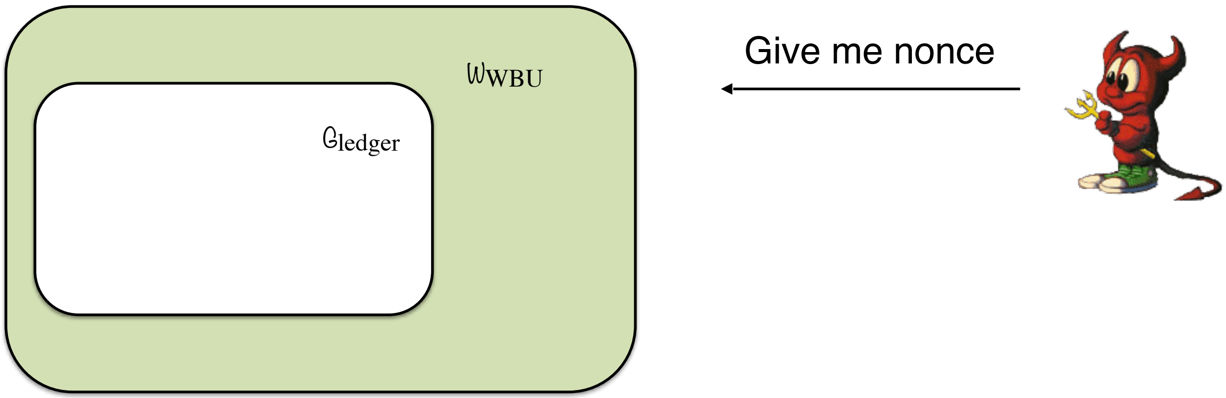

# How can we add randomness to ledger?

## Adding Randomness without changing the ledger



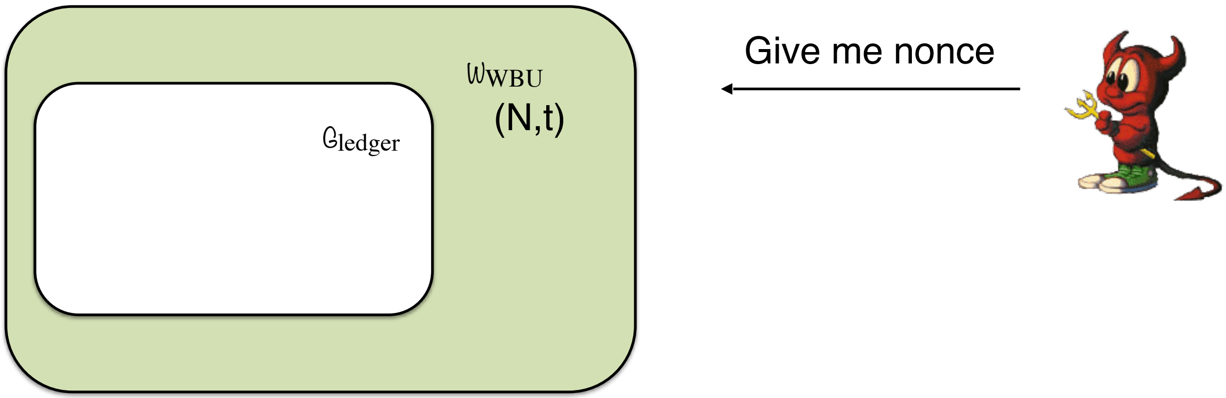

# How can we add randomness to ledger?

## Adding Randomness without changing the ledger



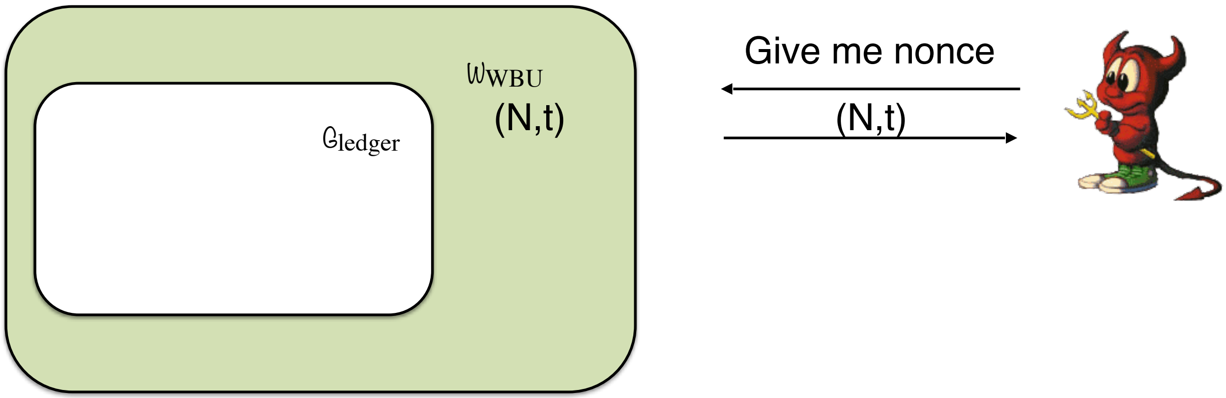
# How can we add randomness to ledger?

## Adding Randomness without changing the ledger



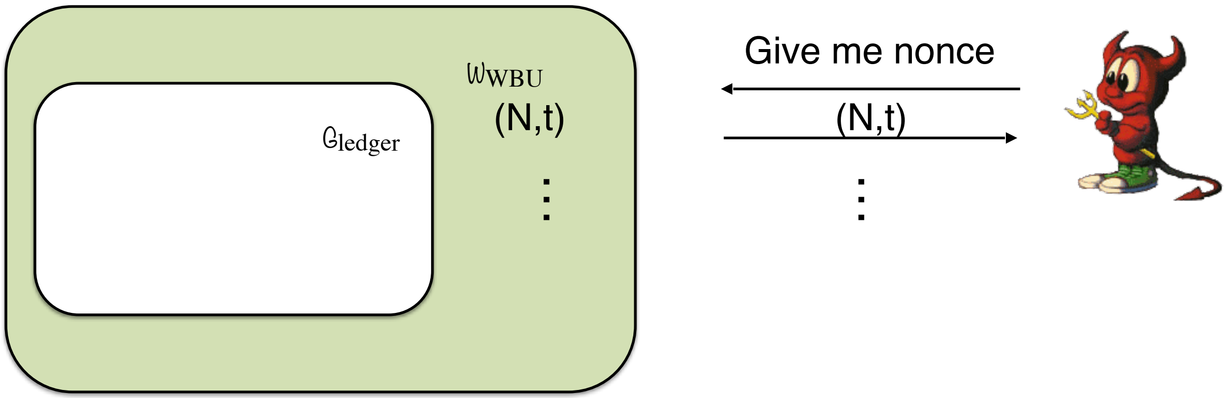
# How can we add randomness to ledger?

## Adding Randomness without changing the ledger



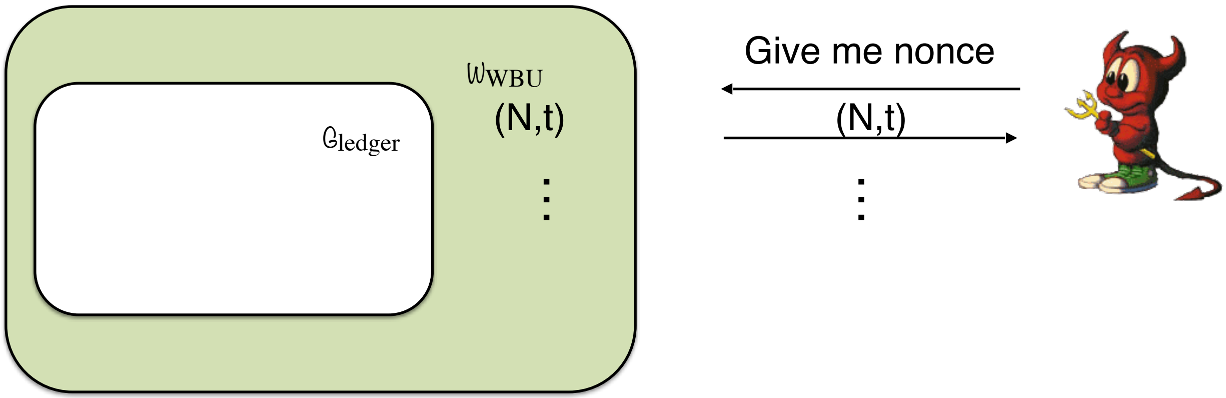
# How can we add randomness to ledger?

## Adding Randomness without changing the ledger



# How can we add randomness to ledger?

## Adding Randomness without changing the ledger

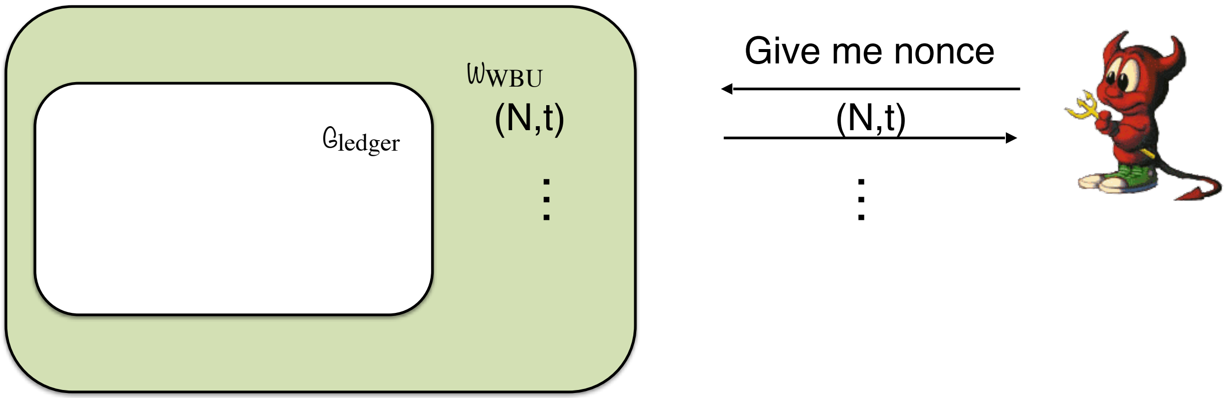



Within a predefined time window δ:

- The adversary needs to insert a block whose nonce was issued within this window by the wrapper
- The window depends on chain growth, chain quality, and network delay

# How can we add randomness to ledger?

## Adding Randomness without changing the ledger



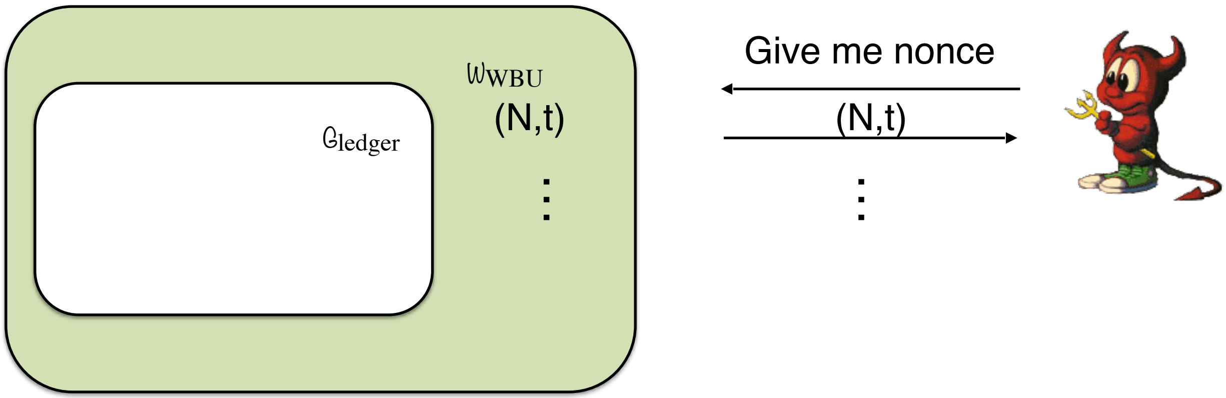

Within a predefined time window δ:

- The adversary needs to insert a block whose nonce was issued within this window by the wrapper
- The window depends on chain growth, chain quality, and network delay

Every δ-long window has a block that the adversary cannot predict before that window

# How can we add randomness to ledger?

## Adding Randomness without changing the ledger




Within a predefined time window δ:

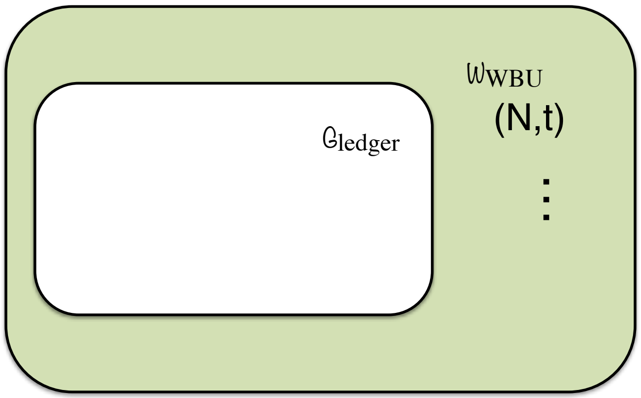
- The adversary needs to insert a block whose nonce was issued within this window by the wrapper
- The window depends on chain growth, chain quality, and network delay

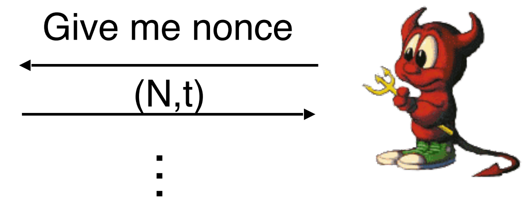
***Weak Beacon** [ACKZ19]*

Every δ-long window has a block that the adversary cannot predict before that window

# How can we add randomness to ledger?

**Adding Randomness without changing the ledger**




Bitcoin implements the wrapped ledger [ACKZ19]

Within a predefined time window δ:

- The adversary needs to insert a block whose nonce was issued within this window by the wrapper
- The window depends on chain growth, chain quality, and network delay

***Weak Beacon*** *[ACKZ19]*

Every δ-long window has a block that the adversary cannot predict before that window

# Summary

- UC definitions of postdate, backdate and timed security for the notions of: signature, ZK and Signature of Knowledge

- Definition and construction of a weak-beacon in the **strong** ledger-hybrid model

Thank you