BOLT: Anonymous Payment Channels for Decentralized Currencies

Paper authors: Matthew Green and Ian Miers

Presentation by: Charlie Liu

What is BOLT?

- B – Blind
- O – Off-chain
- L – Lightweight
- T - Transactions

So what are on-chain transactions?

- Any normal transaction as of right now is considered “on-chain”
- This means the transactions that occurred are broadcasted to the network for miners to confirm
- Issues dealing with basic transactions:
  - Speed – normal procedures involve waiting for confirmations by miners through proof of work; usually around ~10 minutes; 7 transactions processed per second as compared to Visa’s (24,000 per second)
  - “Privacy & Anonymity” - Pseudo-anonymous on the blockchain; can be tracked; all transactions published onto the blockchain

What are Off-chain transactions?

- Transactions that are not broadcasted to the network to be confirmed
- Useful for payment channels
- Off-chain transactions solves the on-chain issues:
  - Speed – off-chain transactions can be recorded immediately; once finished, the off-chain transactions are released with priority for the miners to confirm; lessens workload of the miners
  - This makes it “lightweight”
- Issues that still occur:
  - “Privacy & Anonymity” - depends on the system
    - Some networks such as Lightning Network is not very private within the channel

Privacy and Anonymity Solutions?

- BOLT (paper was made independently of Rayo and Fulgor)

BOLT

- Composed of 3 different channel types
  - Has its own protocol
- Uses Blind signatures, commitments, and “temporary coins” to keep users anonymous and private from each other

Digital Signatures

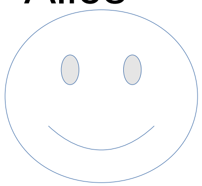
- KeyPair(pk, sk)
- pk unlocks a message encrypted by sk
- sk unlocks a message encrypted by pk

Blind Signatures

- The signer signs a message that is blinded (he or she does not know the contents are), but the message (with the signer's signature) can be later revealed by a third party to confirm that it has been signed by the signer
- This technique is heavily used in BOLT where:
  - Signature requester – customer
  - Signer – merchant
  - 3rd party that verifies – network
- Privacy and anonymity:
  - Signer does not know what he or she is signing so privacy for requester
  - Signer can not link the blinded message to the later revealed message
  - Message can be verified with the signer's public key

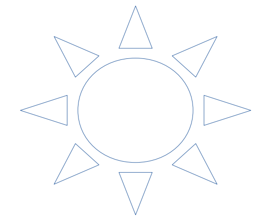
Blind Signature Math

Alice

1. 

e_B, n_B $\longleftarrow$

Bob



sk:

sig

2. Select a random: “r”

   Only Alice knows r

$\sim m = r^e * m \bmod n$ $\longrightarrow$

Alice does not want Bob to know what her message contains, just his signature so,

Alice generates the blinded message:
$\sim m = r^e * m \bmod n$

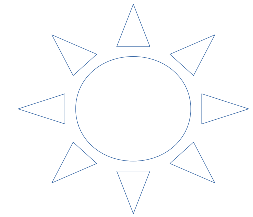
Blind Signature Math

Alice

1.

~S

Bob



sk:

sig

Bob signs the hidden message:
~S = signed (hidden)
~S = $(\sim m)^{sig}$ mod n

2. Obtains S(m) by:

(~S)/r mod n

Because:

~S = $(\sim m)^{sig}$ mod n

~m = $(r^e * m)$ mod n -so: ~S = $([r^e * m] \bmod n)^{sig}$ mod n

$(r^{e*sig} * m^{sig})$ mod n

$r^1 * m^{sig}$ mod n $\rightarrow$ $\frac{r * m^{sig}}{r}$ $\rightarrow$ m^{sig}

e*sig = 1

Pedersen's Commitment Scheme

- Will explain on board

Types of Channels: Unidirectional

- This means that money flows 1 way
- Typically Customer → Merchant
- More for fixed amount of money
- Based off of the **Compact e-cash**

What is the Compact E-Cash?

- Customer creates a temporary fixed-sized wallet (wCom) based on a tuple(*k*, *sk*, *B*) that can generate B number of coins
- The wallet is to be signed by the merchant to confirm validity
  - *k* = seed for pseudo random function F (basically initial value for a random function; pseudo randomness depends on its starting value)
  - *sk* = the customer's private key
  - *B* = the number of coins in the wallet
- Each coin has its tuple (*s*, *T*, *π*)
  - *s* = serial number generated by the pseudo random function $F_k(i)$; i is the current coin
  - *T* = double spend tag
  - *π*= interactive – zero knowledge proof

1. $0 < i \leq B$
2. The prover knows sk.
3. The prover has a signature on the wallet (k, sk, B).
4. The pair (s, T) is correctly structured with respect to the signed wallet.

Compact E-cash continued

- The idea is to label each spent coin/transaction with a serial number in the event that double spending occurs, it can be checked to see if the current serial number is found on a previous coin/transaction

Unidirectional version of Compact E-Cash

- Implements a form of the Compact E-cash protocol with some minor edits
- Customer makes ciphertexts of the coin tuples that they are going to use as from their actual wallet (Not reveal them yet)
  - Merchant does not need to know what these contain, just that they are valid and that the customer do own them

- Merchant signs the temporary wallet on Establish protocol rather than Init protocol

Unidirectional Scheme

$\mathsf{Setup}(1^\lambda)$. On input λ, optionally generate CRS parameters for (1) a secure commitment scheme and (2) a non-interactive zero knowledge proof system. Output these as pp.

$\mathsf{KeyGen}(\mathsf{pp})$. Compute $(pk, sk) \leftarrow \Pi_{\mathsf{sig}}.\mathsf{SigKeygen}(1^\lambda)$.[7]

$\mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_c, sk_c)$. On input a keypair (pk_c, sk_c), uniformly sample two distinct PRF seeds k_1, k_2 and random coins r for the commitment scheme. Compute $\mathsf{wCom} = \mathsf{Commit}(sk_c, k_1, k_2, B_0^{\mathsf{cust}}; r)$. For $i = 1$ to B, sample $ck_i \leftarrow \mathsf{SymKeyGen}(1^\lambda)$ to form the vector $\vec{ck}$. Output $\mathsf{T}_{\mathcal{C}} = (\mathsf{wCom}, pk_c)$ and $csk_{\mathcal{C}} = (sk_c, k_1, k_2, r, B_0^{\mathsf{cust}}, \vec{ck})$.

$\mathsf{Init}_{\mathcal{M}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_m, sk_m)$. Output $\mathsf{T}_{\mathcal{M}} = pk_m$, $csk_{\mathcal{M}} = (sk_m, B_0^{\mathsf{cust}})$.

$\mathsf{Refund}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}}, w)$. Parse w (generated by the Establish and Pay protocols) to obtain $\vec{ck}$ and the current coin index i. Compute $\sigma \leftarrow \mathsf{Sign}(sk_c, \mathsf{refund} \| \mathsf{cID} \| i \| ck_i)$ (where cID uniquely identifies the channel being closed) and output $\mathsf{rc}_{\mathcal{C}} := (\mathsf{cID}, i, ck_i, \sigma)$.

$\mathsf{Refute}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathbf{S}, \mathsf{rc}_{\mathcal{C}})$. Parse the customer's channel closure message $\mathsf{rc}_{\mathcal{C}}$ as $(\mathsf{cID}, i, ck_i, \sigma)$ and verify cID and the signature σ. If the signature verifies, then obtain the ciphertexts $C_i, \ldots, C_B$ stored after the Establish protocol. For $j = i$ to B, compute $(j \| s_j \| u_j \| \pi_j^r \| ck_j \| \hat{\sigma}_j) \leftarrow \mathsf{SymDec}(ck_j, C_j)$ and verify the signature $\hat{\sigma}_j$ and the proof π_j^r. If (1) the signature $\hat{\sigma}_j$ or the proof π_j^r fail to verify, (2) any ciphertext fails to decrypt correctly, or (3) any of the decrypted values (s_j, u_j) match a valid spend containing (s_j, t_j) in $\mathbf{S}$ where $\mathsf{OTDec}(u_j, t_j) = pk_c$: record the invalid result into $\mathsf{rc}_{\mathcal{M}}$ along with cID and sign the result using sk_m so that it can be verified by the network. Otherwise set $\mathsf{rc}_{\mathcal{M}} = (\mathsf{accept})$ and sign with sk_m. Finally for each valid C_j, set $\mathbf{S} \leftarrow \mathbf{S} \cup (s_j, t_b, \pi)$ and output $\mathbf{S}$ as the new merchant state.

Unidirectional Scheme : Setup the Commitment Scheme

up(1^λ). On input λ, optionally generate CRS parameters for (1) a secure commitment scher
and (2) a non-interactive zero knowledge proof system. Output these as pp.

- Uses Pedersen's commitment scheme that is published as pp (public parameter – something both parties can access)
  - This generates 2 things:
    - A CRS (common reference string) between the two parties
    - A non interactive zero – knowledge proof system
- I will explain this on the board:

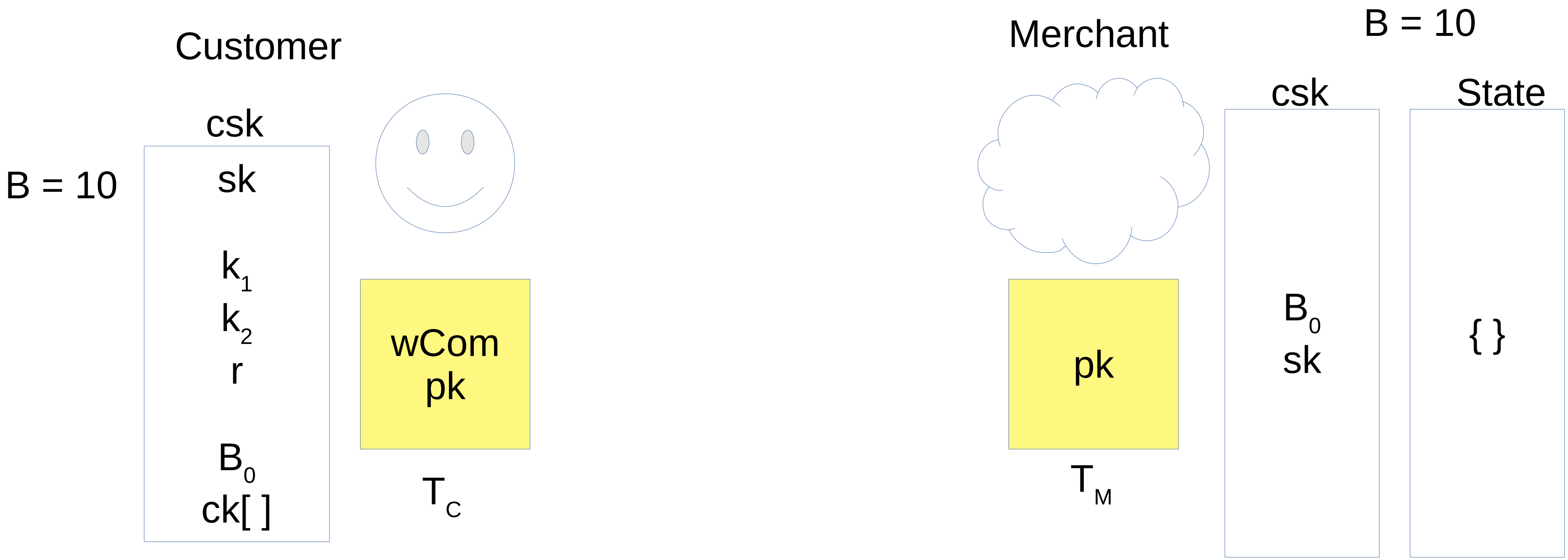
Unidirectional Scheme : KeyGen

yGen(pp). Compute $(pk, sk) \leftarrow \Pi_{\mathsf{sig}}.\mathsf{SigKeygen}(1\hat{})$.'

- Computes a public/private key pair using the commitment earlier for both parties
- Essentially pseudo keys/IDs for both parties

Unidirectional Scheme : Init

p, $B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_c, sk_c)$. On input a keypair (pk_c, sk_c), uniformly sample two distinct P
eeds k_1, k_2 and random coins r for the commitment scheme. Compute $\mathsf{wCom} = \mathsf{Commit}($
$, k_2, B_0^{\mathsf{cust}}; r)$. For $i = 1$ to B, sample $ck_i \leftarrow \mathsf{SymKeyGen}(1^\lambda)$ to form the vector $\vec{ck}$. Out
$_C = (\mathsf{wCom}, pk_c)$ and $csk_C = (sk_c, k_1, k_2, r, B_0^{\mathsf{cust}}, \vec{ck})$.
pp, $B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_m, sk_m)$. Output $\mathsf{T}_{\mathcal{M}} = pk_m$, $csk_{\mathcal{M}} = (sk_m, B_0^{\mathsf{cust}})$.



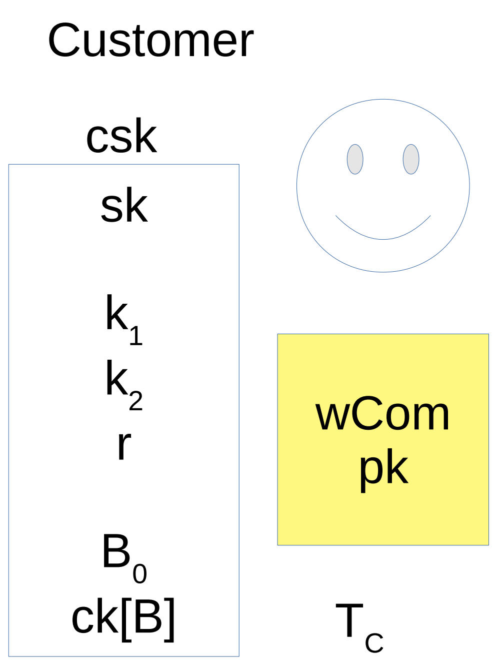
B_0 = customer's initial channel balance

Total = 20 in channel

Unidirectional Scheme : Init

- Both parties will agree on the respective initial channel balances: B_o
- This will be released and verified as an escrow transaction
- A coin vector will be generated from 1 to the total balance

Unidirectional Scheme : Establish



- wCom is your wallet commitment
- This means that you want to prove that the wallet amount you want to use in the channel same as the initial balance you chose
  - You commit is (sk_C, k_1, k_2; r)
- Sign symmetrically encrypted coin tuples

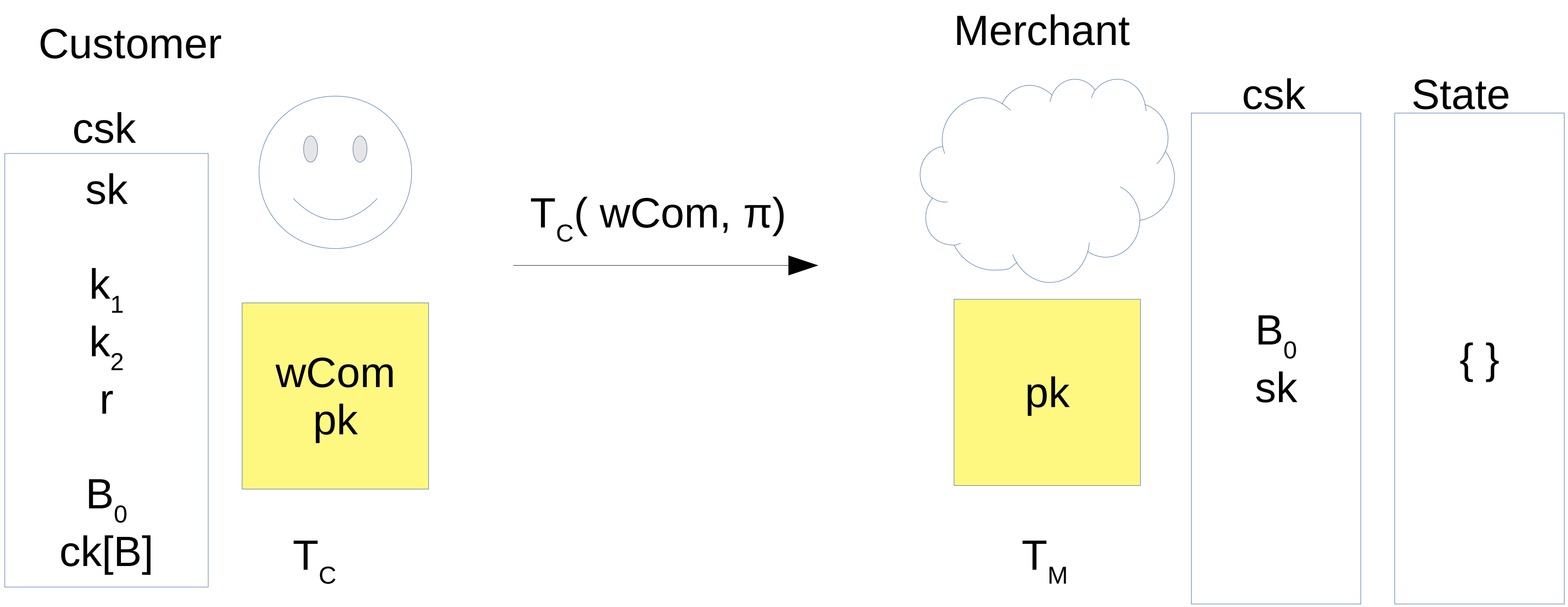
3. For $j = 1$ to B:
   (a) Compute $s_j \leftarrow F_{k_1}(j), u_j \leftarrow F_{k_2}(j), \pi_j^r$ where

$$\begin{aligned}\pi_j^r = PK\{(sk_c, k_1, k_2, r) :\ & s = F_{k_1}(j)\ \wedge u = F_{k_2}(j)\\ & \wedge \mathsf{wCom} = \mathsf{Commit}(sk_c, k_1, k_2; r)\\ & \wedge (pk_c, sk_c) \in \mathsf{KeyGen}(1^\lambda)\}\end{aligned}$$

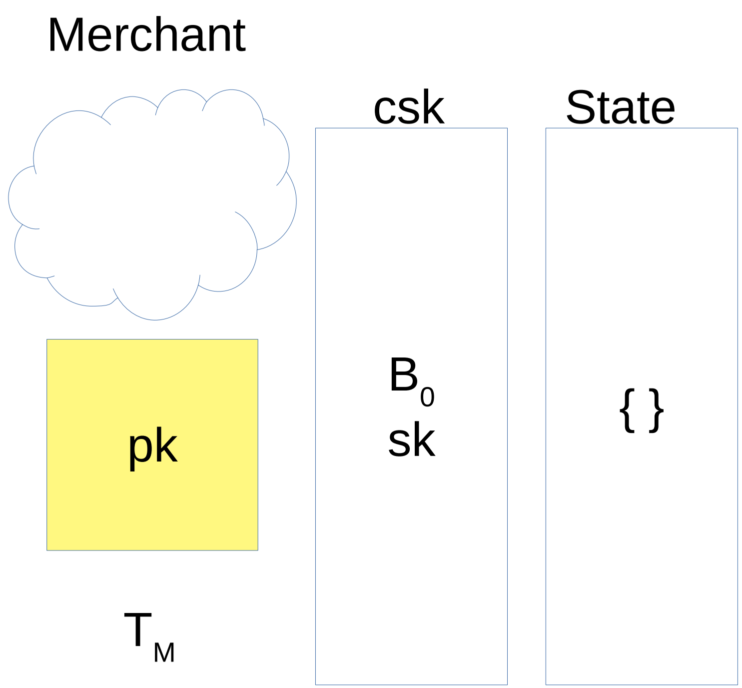
   (b) Compute an internal signature $\hat{\sigma}_j = \mathsf{Sign}(sk_c, \mathsf{spend}\|j\|s_j\|u_j\|\pi_j^r\|ck_{j+1})$.
   (c) Compute $C_j = \mathsf{SymEnc}(ck_j, j\|s_j\|u_j\|\pi_j^r\|\hat{\sigma}_j\|ck_{j+1})$ and

Unidirectional Scheme : Establish

Send proof of valid money that I have so the merchant can approve my channel wallet



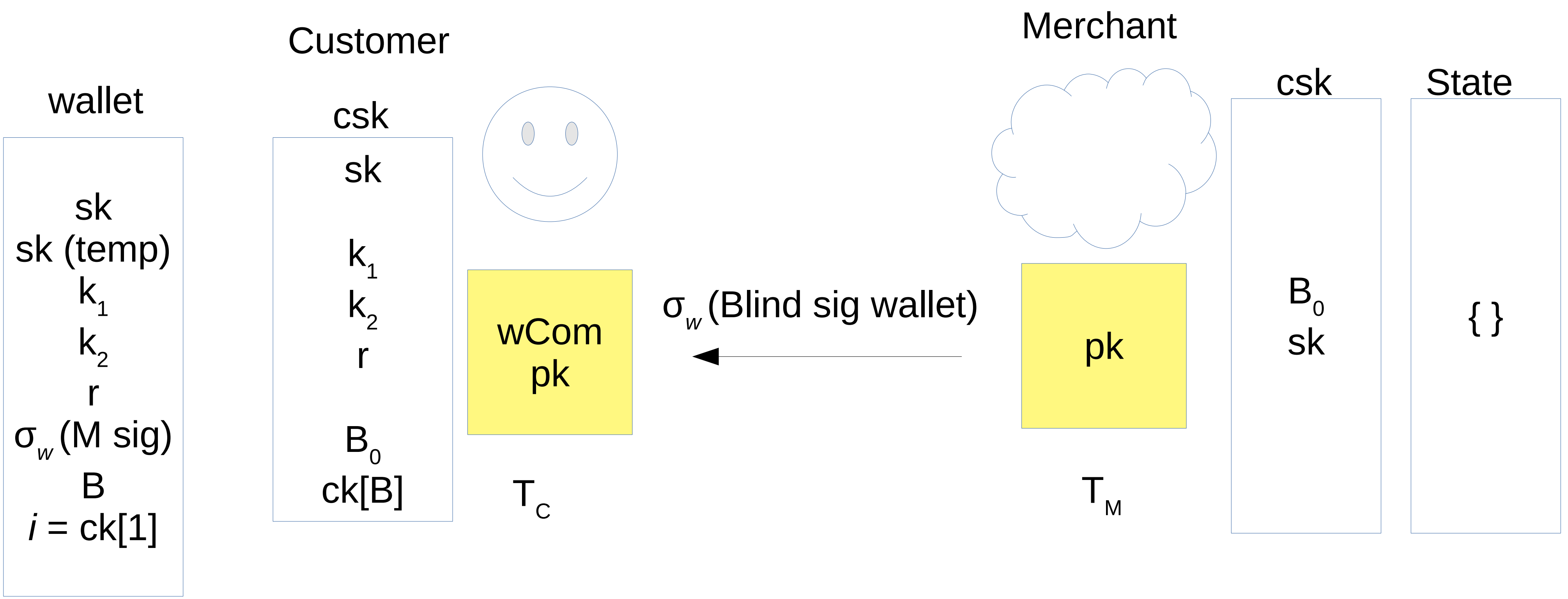
Unidirectional Scheme : Establish



B = # of coins

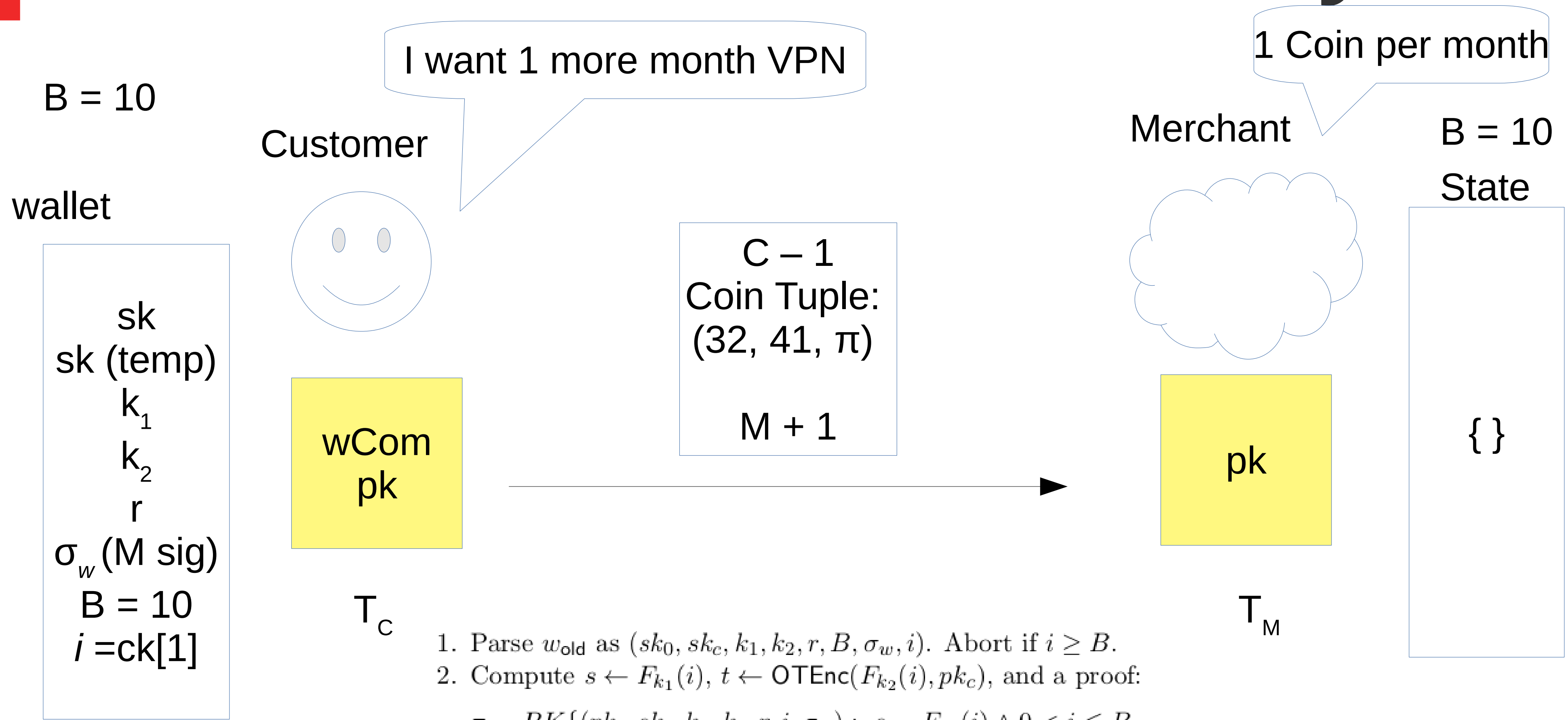
- Takes customer token that contains wCom and its proof (π)
- Verify the customer's signature on Token (T_C) and its proof
- Verify that B_0 = B by checking the j^{th} coin
- Coin encryptions are considered "chained" (I assume this means that unlocking the j^{th} coin will produce the key to decrypt the j^{th} - 1 coin)

Unidirectional Scheme : Establish



- If checks done by the merchant is valid then send the customer's temp wallet with a blind signature signed by the merchant
- Else if something fails then abort channel

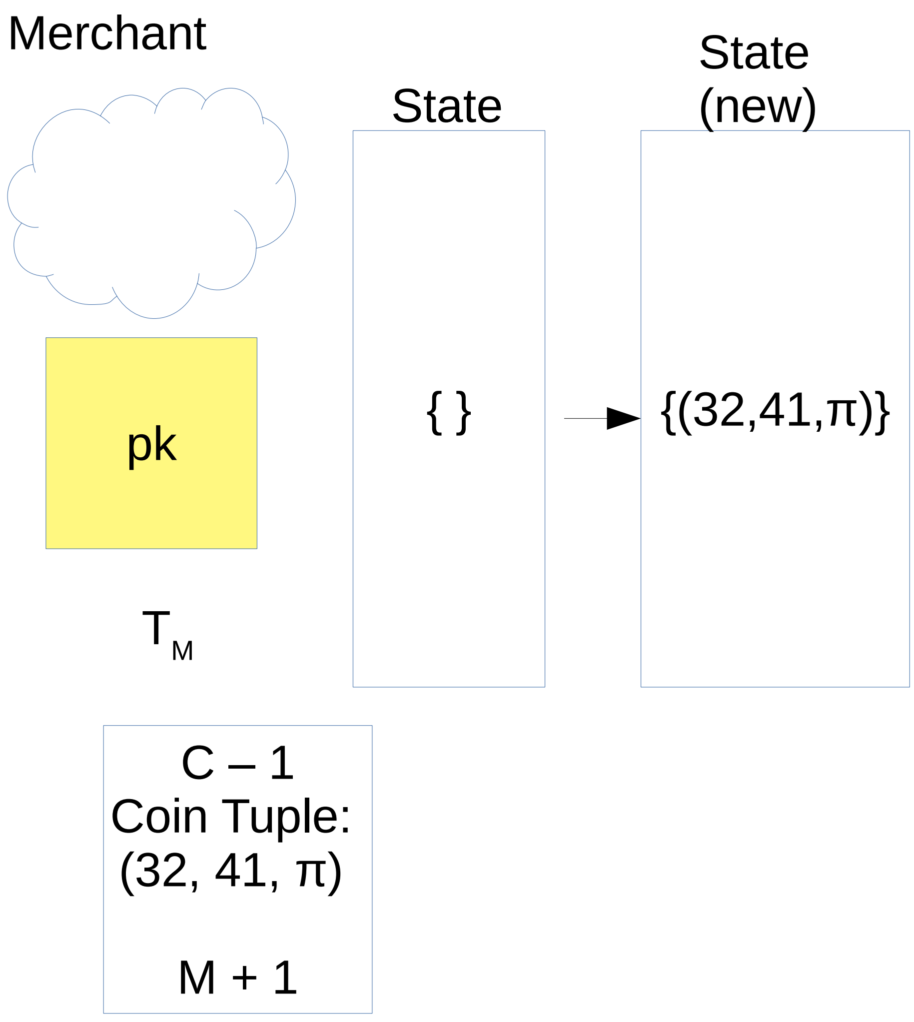
Unidirectional Scheme : Pay



1. Parse w_{old} as $(sk_0, sk_c, k_1, k_2, r, B, \sigma_w, i)$. Abort if $i \geq B$.
2. Compute $s \leftarrow F_{k_1}(i)$, $t \leftarrow \mathsf{OTEnc}(F_{k_2}(i), pk_c)$, and a proof:

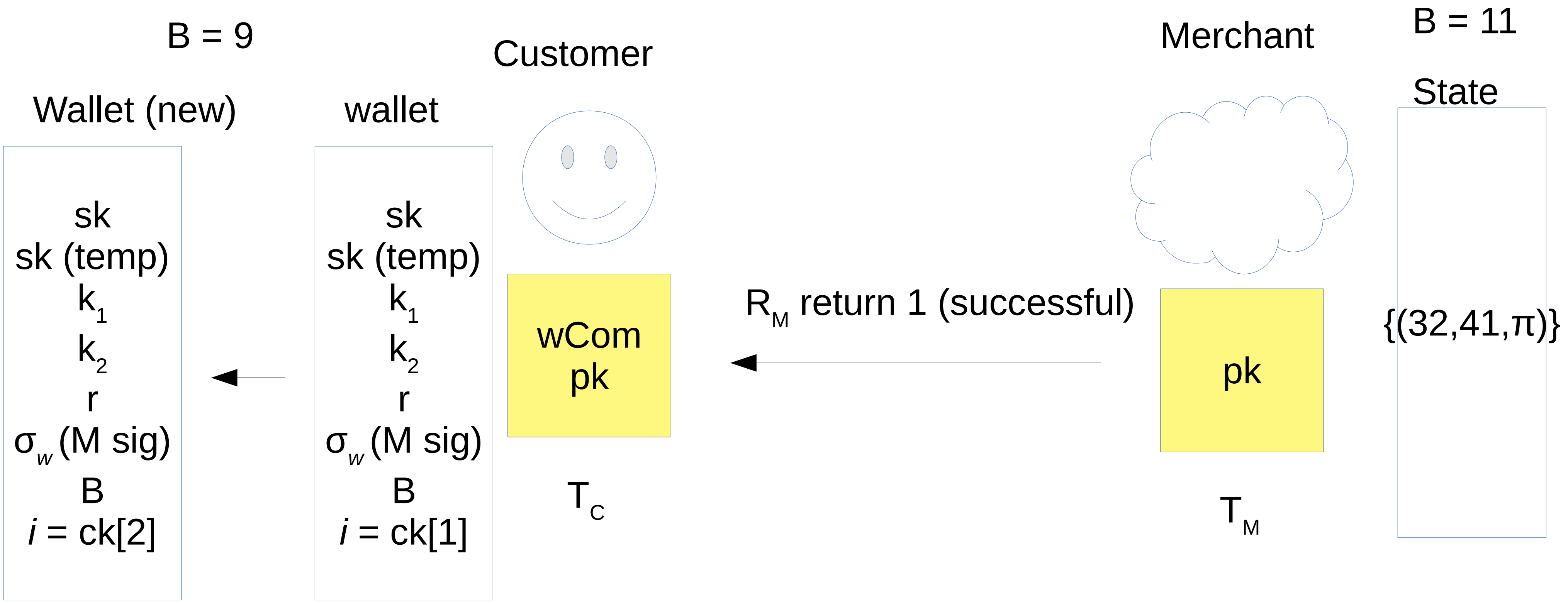
$$\begin{aligned}\pi = PK\{(pk_c, sk_c, k_1, k_2, r, i, \sigma_w) :\ & s = F_{k_1}(i) \wedge 0 < i \leq B \\ & \wedge\ t = \mathsf{OTEnc}(F_{k_2}(i), pk_c) \\ & \wedge\ \mathsf{Verify}(pk_m, (k_1, k_2, sk_c), \sigma_w) \\ & \wedge\ (pk_c, sk_c) \in \mathsf{KeyGen}(\mathsf{pp})\ \}\end{aligned}$$

Unidirectional Scheme : Pay

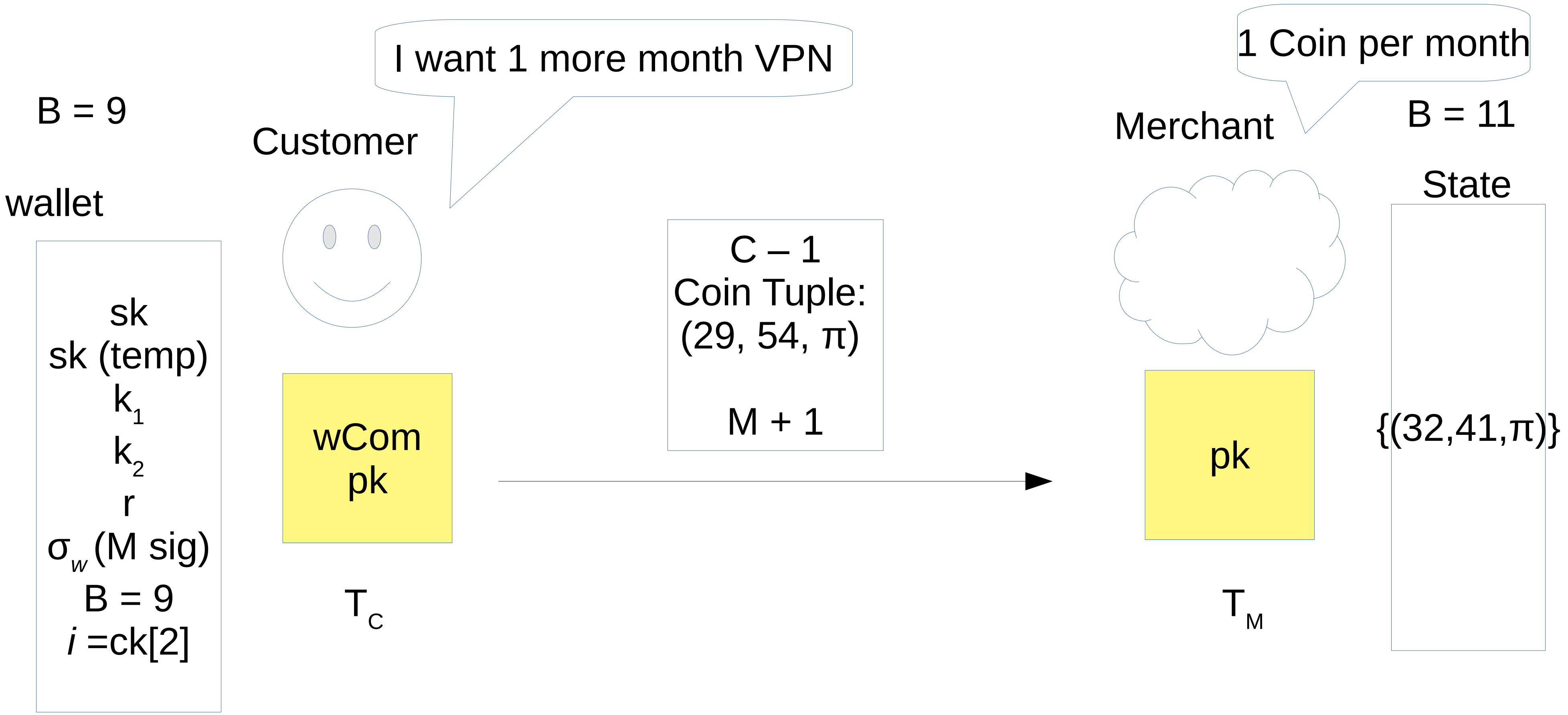


- Verifies π and make sure that (s,.,.) is not in State
- In this example, State was initially empty and *s* in coin tuple is “32”
- State then stores the coin tuple
- Merchant’s R_M returns 1 (true)
- R_M = payment success bit (boolean)

Unidirectional Scheme : Pay

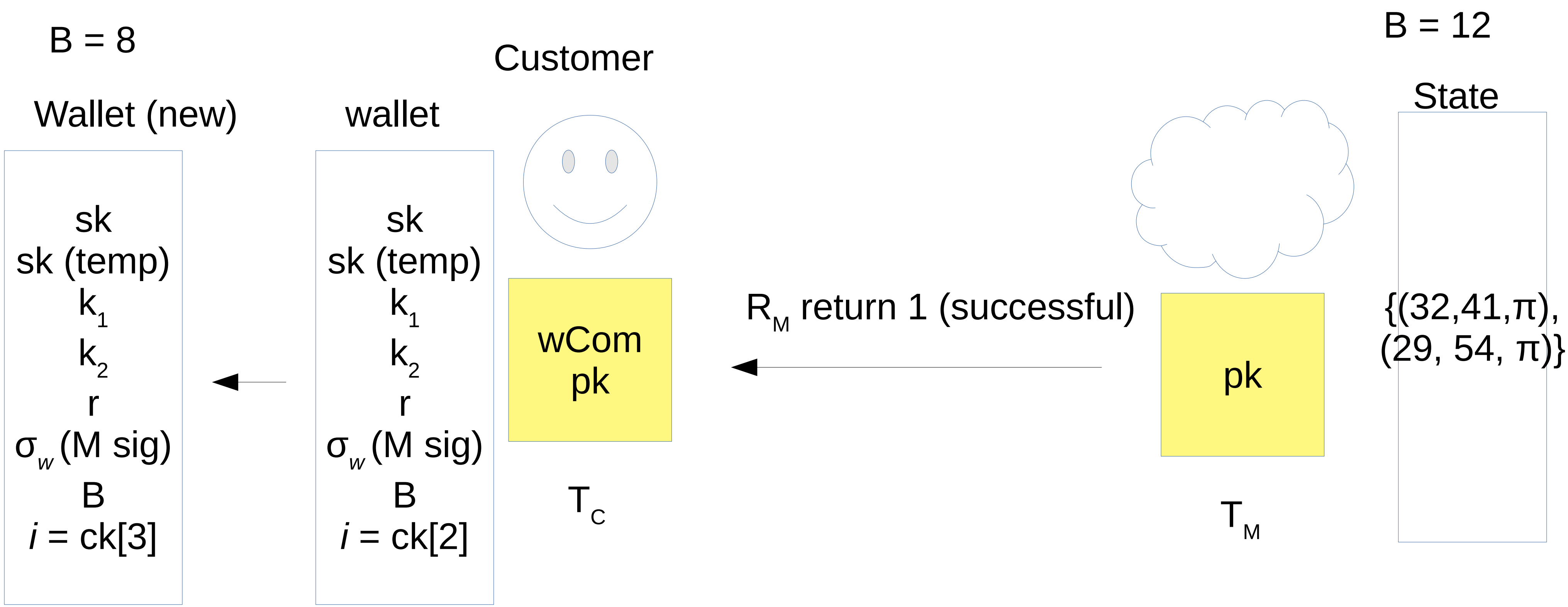


- Coin vector index ++ (essentially means make the next coin tuple the valid one)
- Makes it very difficult to double spend



I want 1 more month VPN
1 Coin per month
B = 9
Customer
Merchant
B = 11
wallet
State
sk
sk (temp)
k_1
k_2
r
σ_w (M sig)
B = 9
i =ck[2]
wCom
pk
T_C
C – 1
Coin Tuple:
(29, 54, π)
M + 1
pk
T_M
{(32,41,π)}

Unidirectional Scheme : Pay



Unidirectional Scheme : Closure

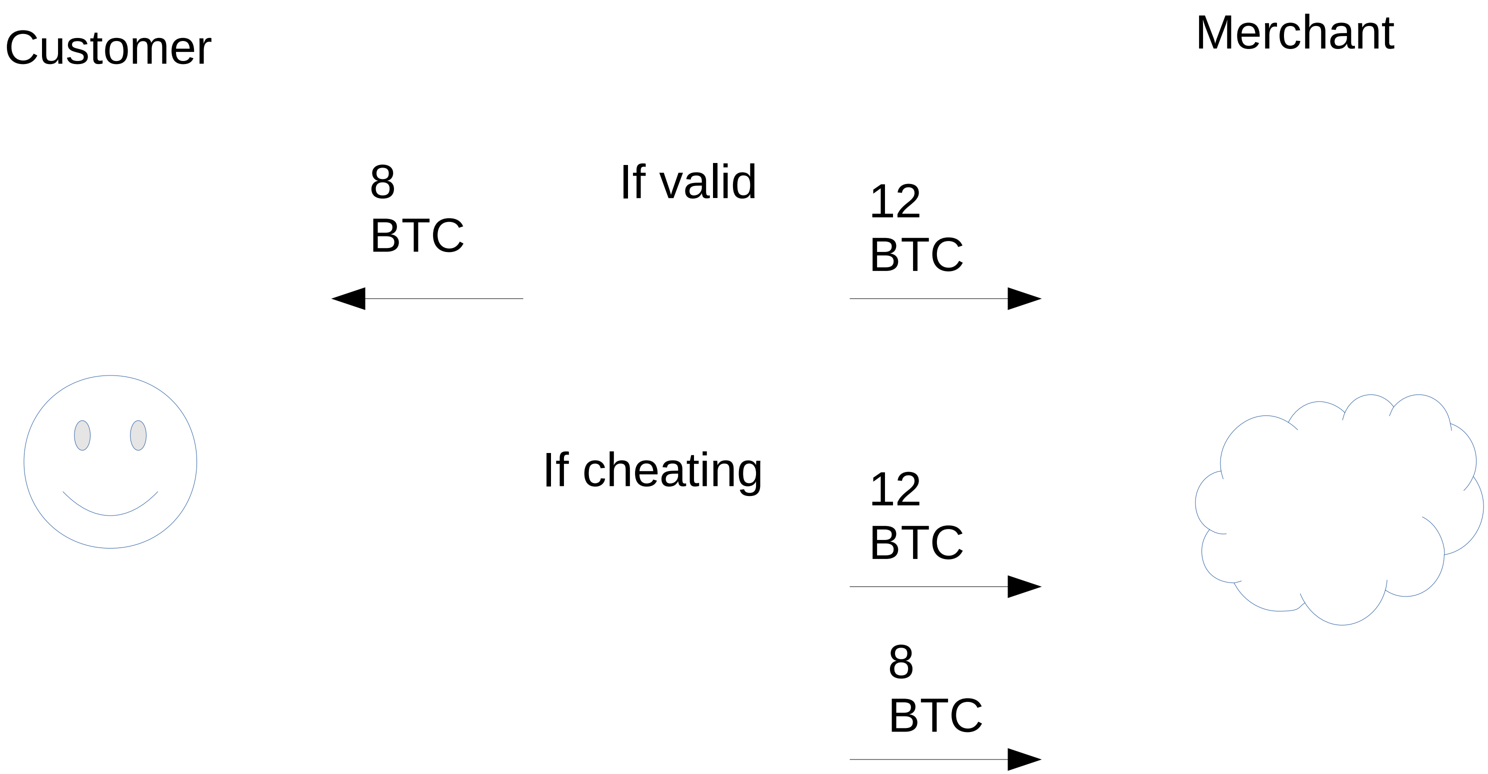
- Both parties initiate their own protocol for closure
- Customer runs Refund protocol to obtain the current coin index (current I) and request a rc_M (merchant's channel closure message) from the merchant
  - This protocol outputs a rc_C (customer's channel closure message) that contains the channel closure ID, current coin index, the coin at the current coin index, and the customer's signature
- Merchant responds with its Refute protocol
  - The merchant then verifies the signature, the unspent coins in the coin vector, and make sure that the same unspent coins are in the merchant's State
  - If valid, sign own rc_M, accept customer's rc_C, and make sure the fake/temp money matches the ciphertext money sent in the beginning

Cheating?

- During channel closure verification by the merchant, if any of the verification fails:
  - Signature of the coins
  - Proof of the coins
  - Unable to decrypt ciphertexts
  - Or double spending occurs
- Merchant records the proof in its rc_M, signs it and lets the network confirm

Unidirectional Scheme : Resolve

- Resolve protocol is now ran by the network to determine the final channel balance and determines the money sent to each party



Channel Type : Bidirectional

- No longer giving coins
- Transactions in both ways: Customer ← → Merchant
- With each transaction the customer gets a new wallet that contains *W*(old wallet) – ε (amount exchanged) compared to Unidirectional that just deals with changing the coin amount in 1 wallet
- *Customer closes the channel with a refund token given by the merchant

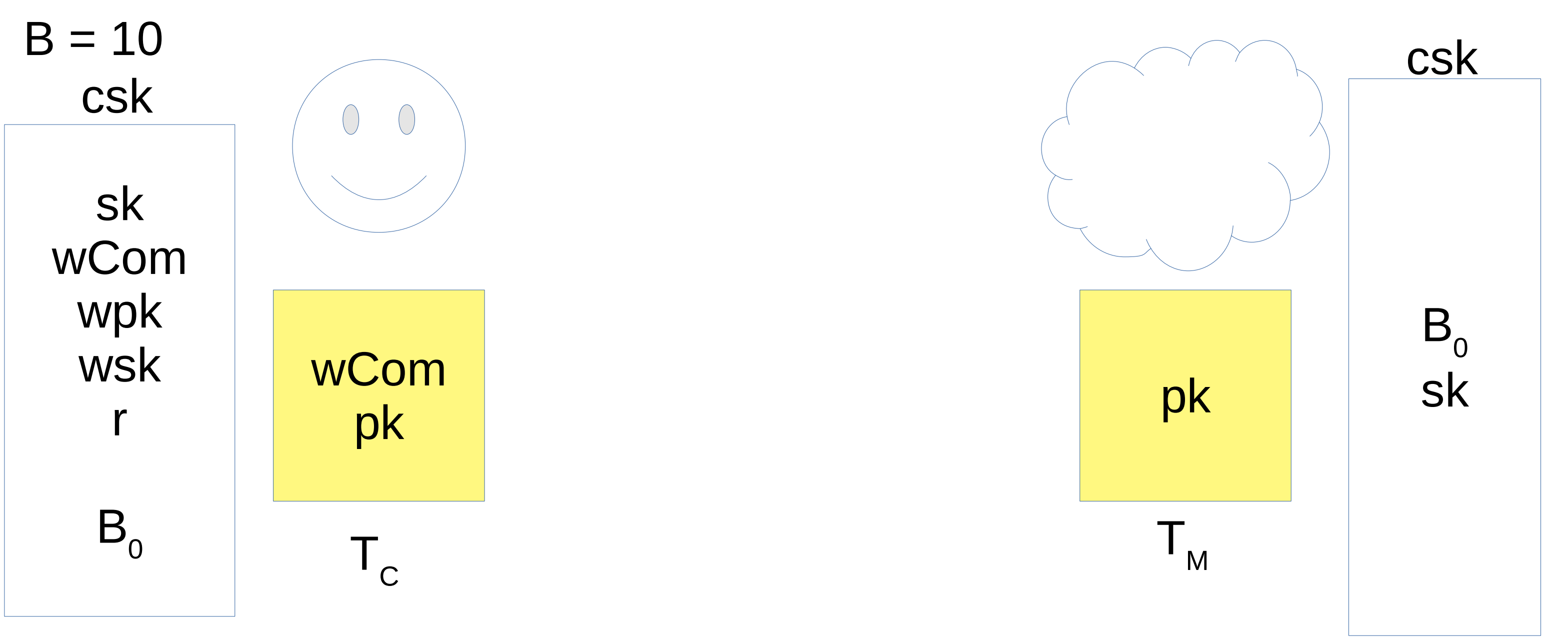
Bidirectional : Init

- Setup's the same

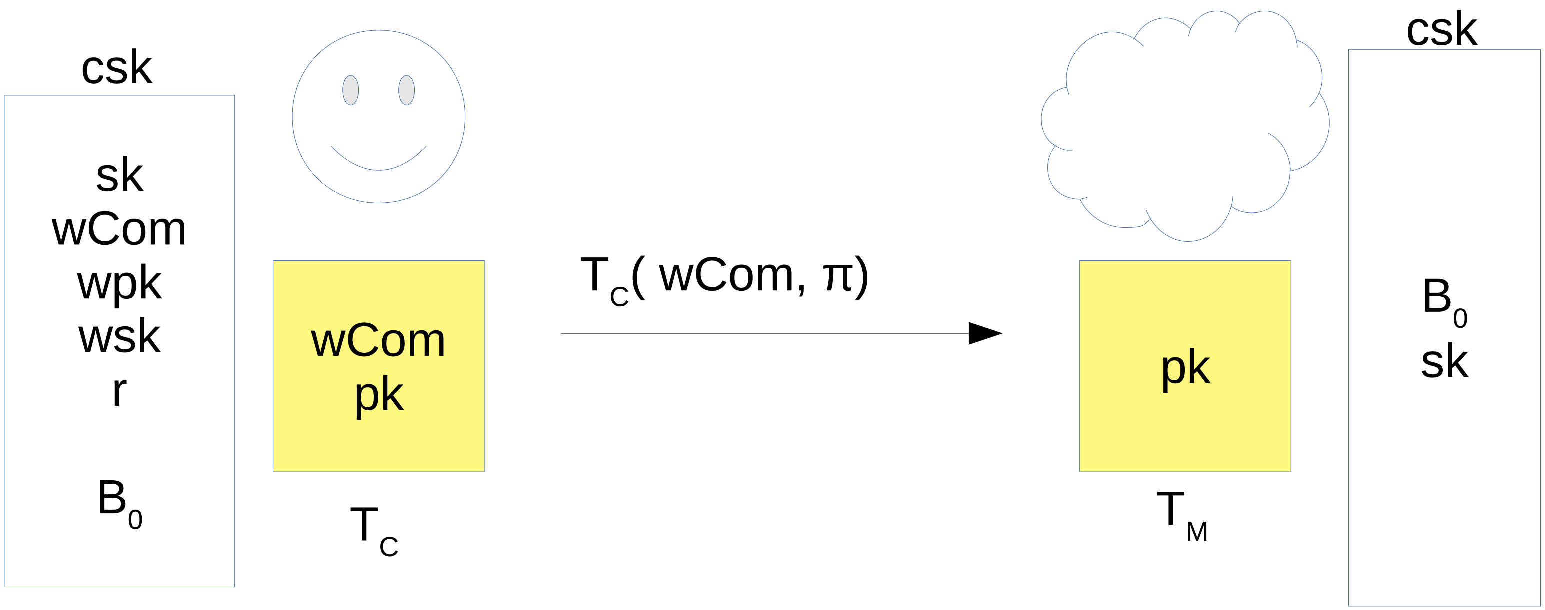
$\mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_c, sk_c)$. The customer generates the wallet commitment by sampling random coins r, computing an ephemeral keypair $(wpk, wsk) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and producing a commitment $\mathsf{wCom} = \mathsf{Commit}(wpk, B_0^{\mathsf{cust}}; r)$. It outputs:

$$\mathsf{T}_{\mathcal{C}} = (pk_c, \mathsf{wCom}) \qquad csk_{\mathcal{C}} = (\mathsf{wCom}, sk_c, wpk, wsk, r, B_0^{\mathsf{cust}})$$

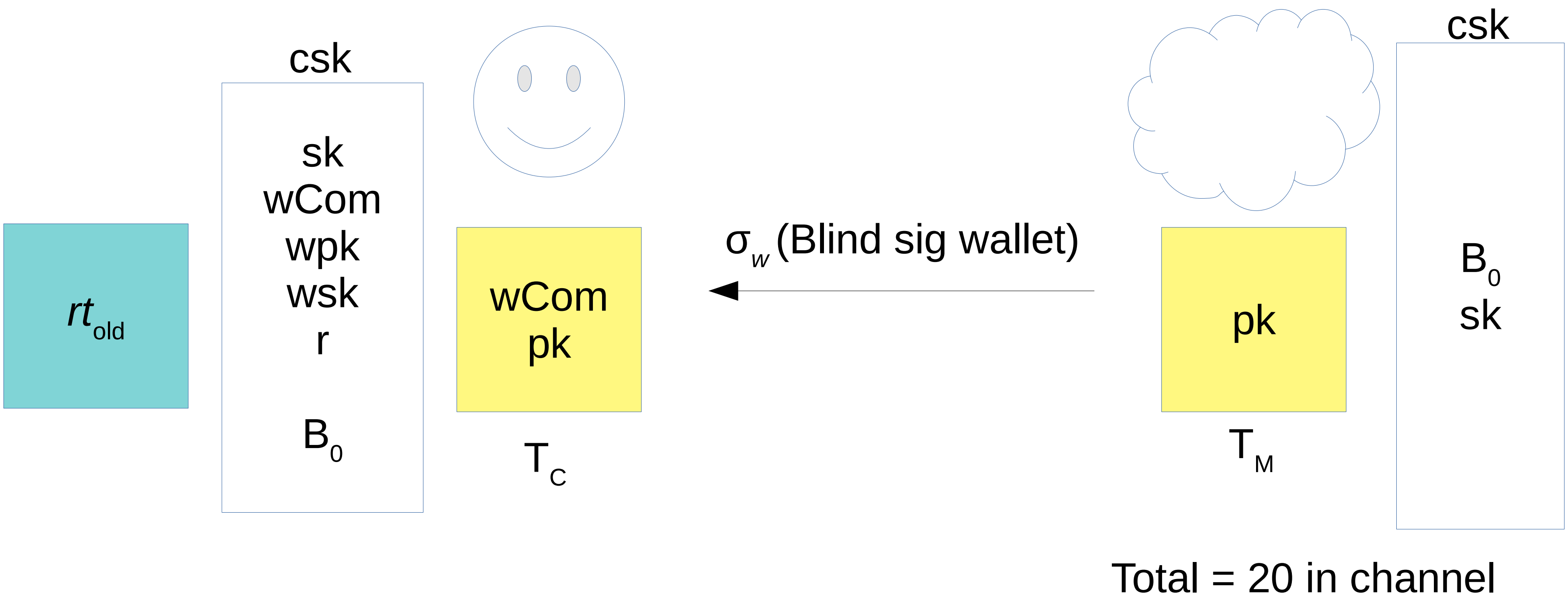
$\mathsf{Init}_{\mathcal{M}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_m, sk_m)$. Output $\mathsf{T}_{\mathcal{M}} = pk_m$, $csk_{\mathcal{M}} = (sk_m, B_0^{\mathsf{cust}})$.



Bidirectional : Establish

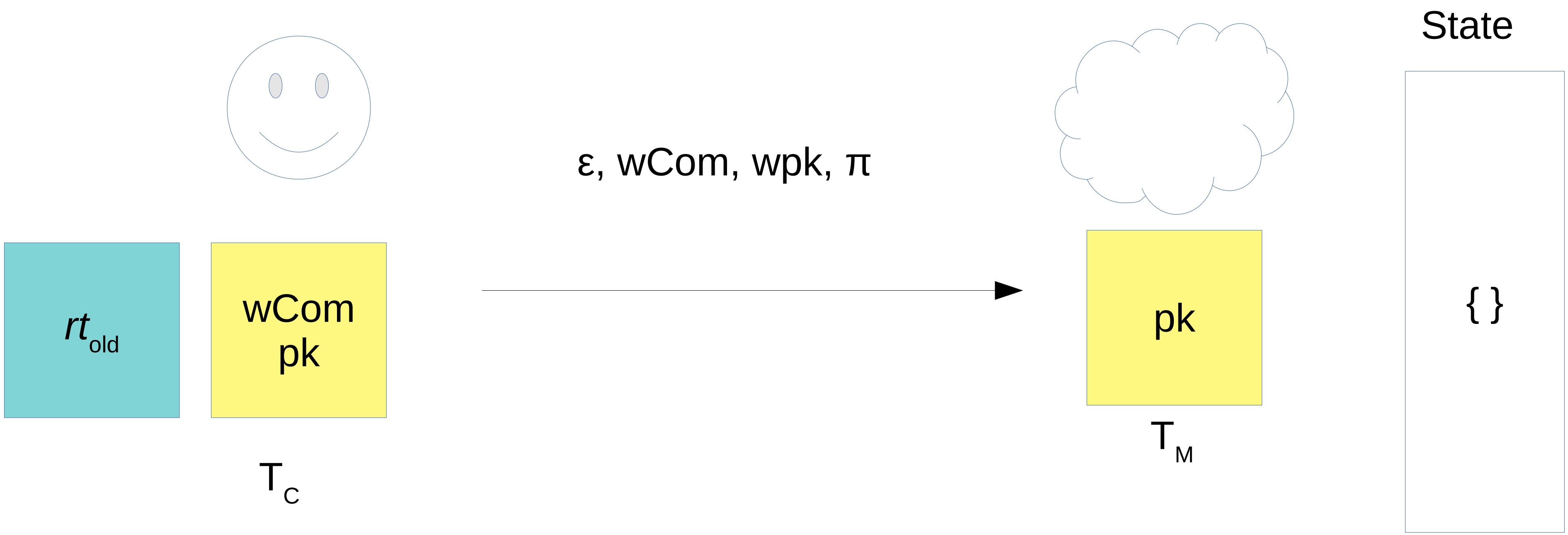


- Pretty much the same except you replace your keys (the ones in uni) with the temporary wallet's keys



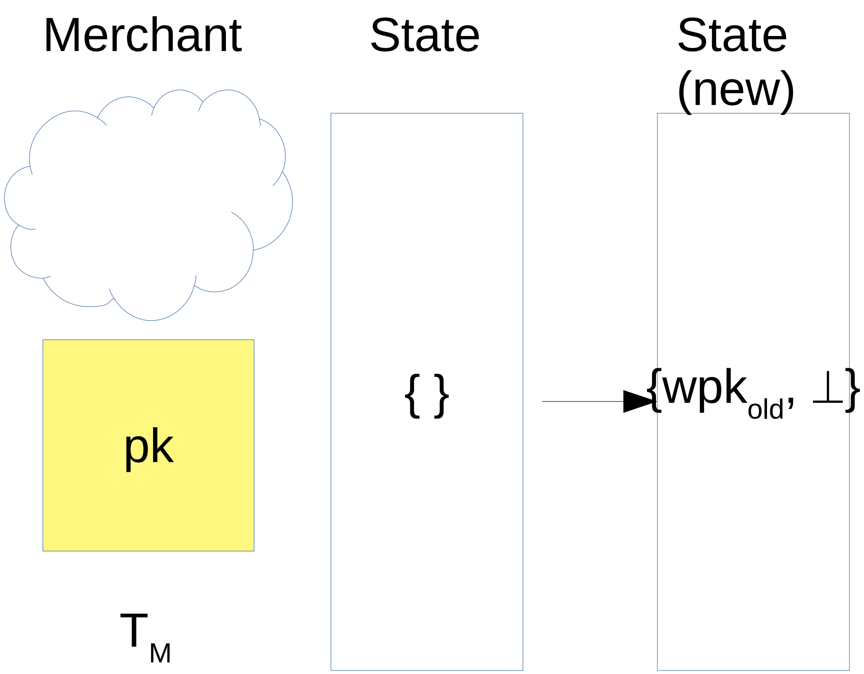
rt_{old}
csk
sk
wCom
wpk
wsk
r
B_0
wCom
pk
T_C
σ_w (Blind sig wallet)
pk
T_M
csk
B_0
sk
Total = 20 in channel

Bidirectional : Pay



- The customer presents the merchant this copy of his old wallet (referenced by the wallet's public key) as well as the amount (ε) that the customer wishes to change in the new wallet by (w_{new} - ε)

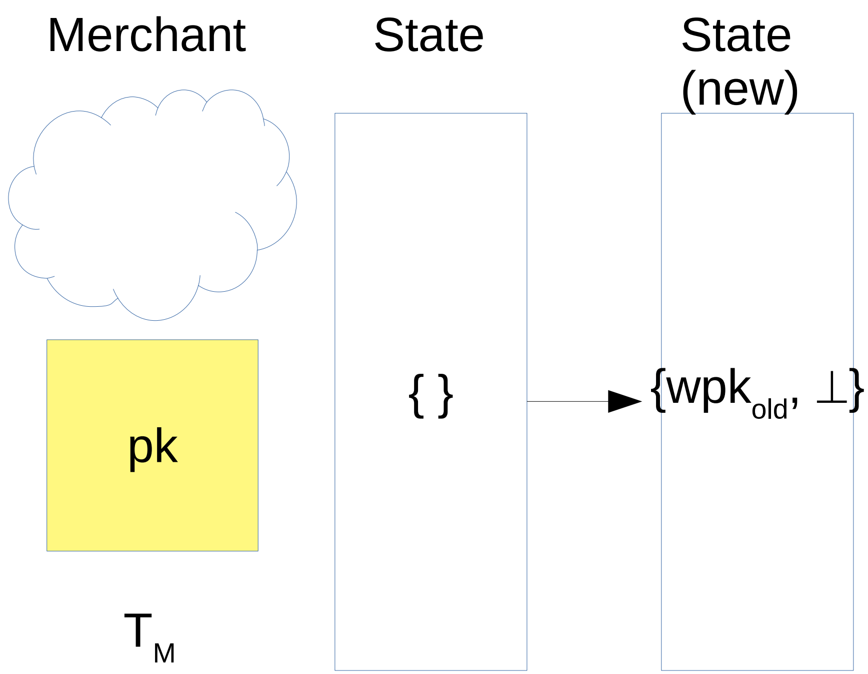
Bidirectional : Pay



- Verify π , and check that the transaction amount fits between 0 and wallet[wallet.size]
- If the below proof is unverifiable/valid, abort the transactions and output ⊥ (failure)
- If the proof is verifiable, add the old wallet to your State and set it to ⊥ (void in this case)
  - This really just means voiding the customer's old wallet in your records

$$\pi_2 = PK\{(wpk', B, r', \sigma_w) : \ \mathsf{wCom}' = \mathsf{Commit}(wpk', B - \epsilon; r') \\ \wedge \mathsf{Verify}(pk_m, (wpk, B), \sigma_w) = 1 \\ \wedge 0 \leq (B - \epsilon) \leq \mathsf{val}_{\mathsf{max}} \ \}$$

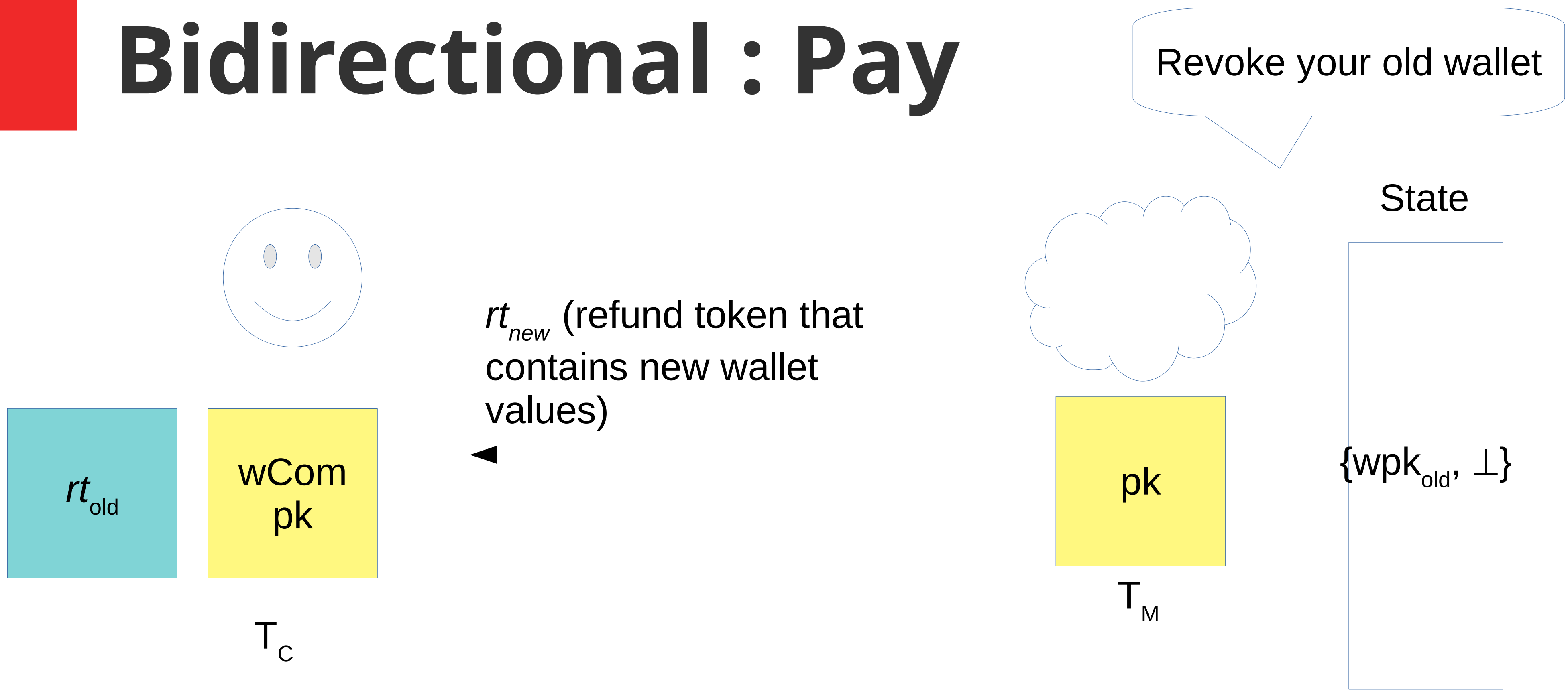
Bidirectional : Pay



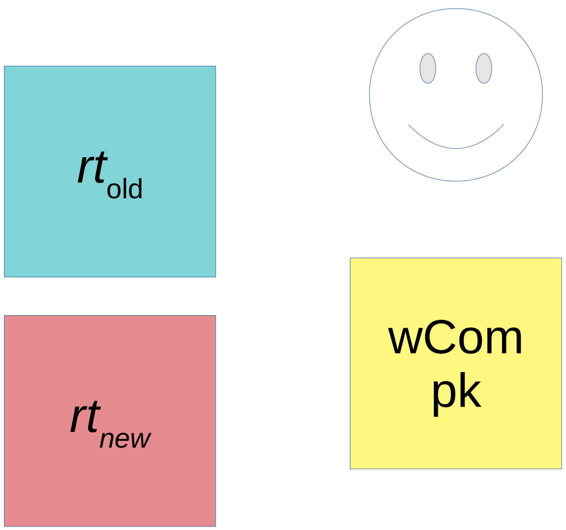
- Merchant sends the customer a refund token denoted: rt_{new} that contains a partially blind signature of the merchant on the new requested wallet

with the customer to provide a partially blind signature $rt_{w'}$ under sk_m on the message $(\mathsf{refund}\|wpk'\|B-\epsilon)$, where wpk' and $B-\epsilon$ are the contents of wCom'.

Bidirectional : Pay



Bidirectional : Pay

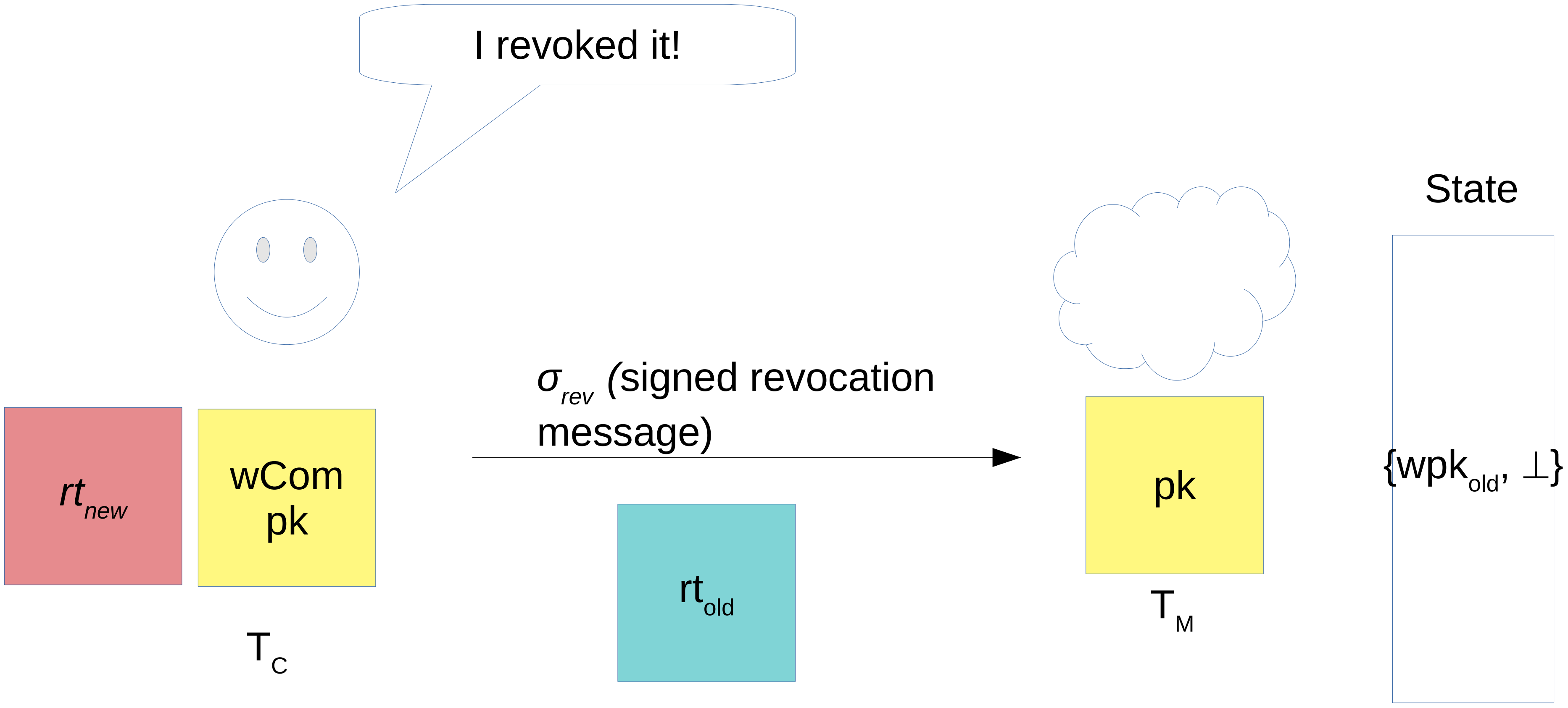


- Customer takes the rt_w and verify that it is indeed from the merchant
- If the verification is a success, the customer invalidates the old wallet by sending the merchant a signed revocation token

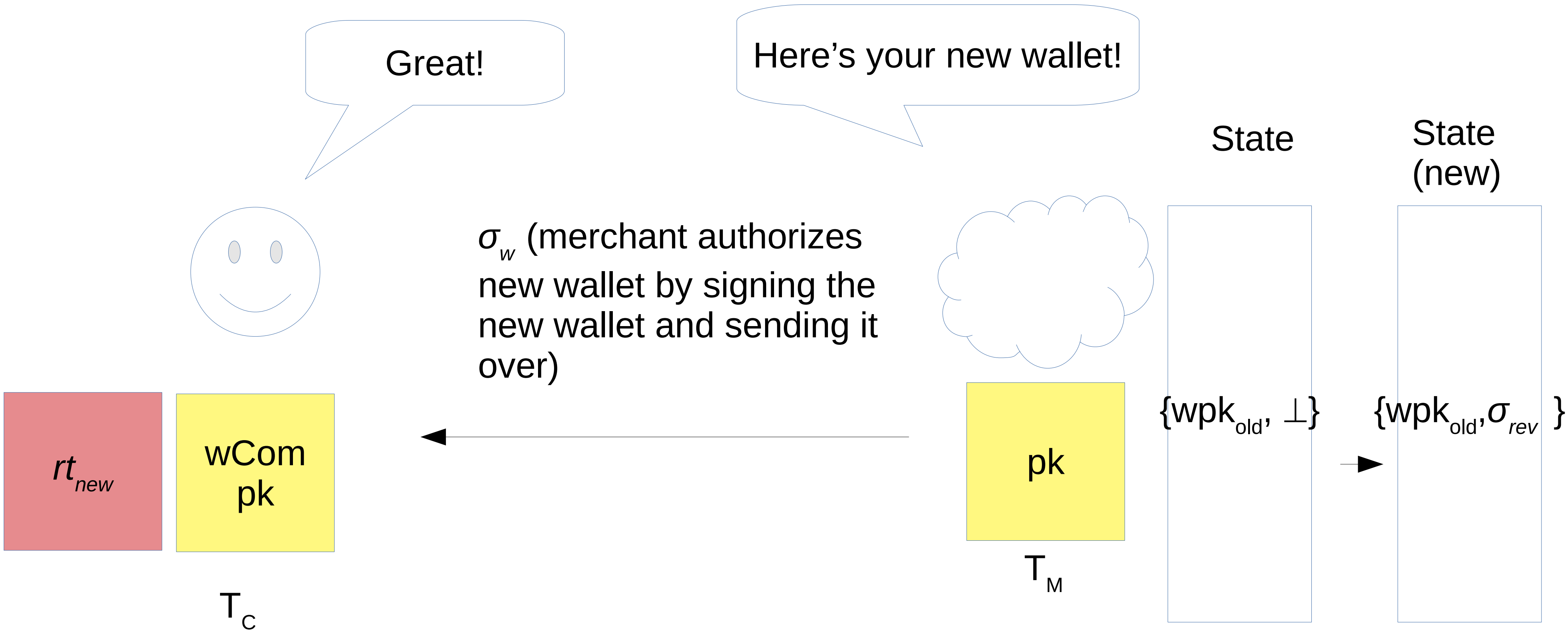
Compute $\mathsf{Verify}(pk_m, rt_{w'}, \mathsf{refund}\|wpk'\|B-\epsilon)$. If verification fails, or if this message does not arrive, abort and output $rt_{w'}$. Else compute $\sigma_{rev} = \mathsf{Sign}(wsk, \mathsf{revoke}\|wpk)$.

$$\xrightarrow{\quad \sigma_{rev} \quad}$$

Bidirectional : Pay

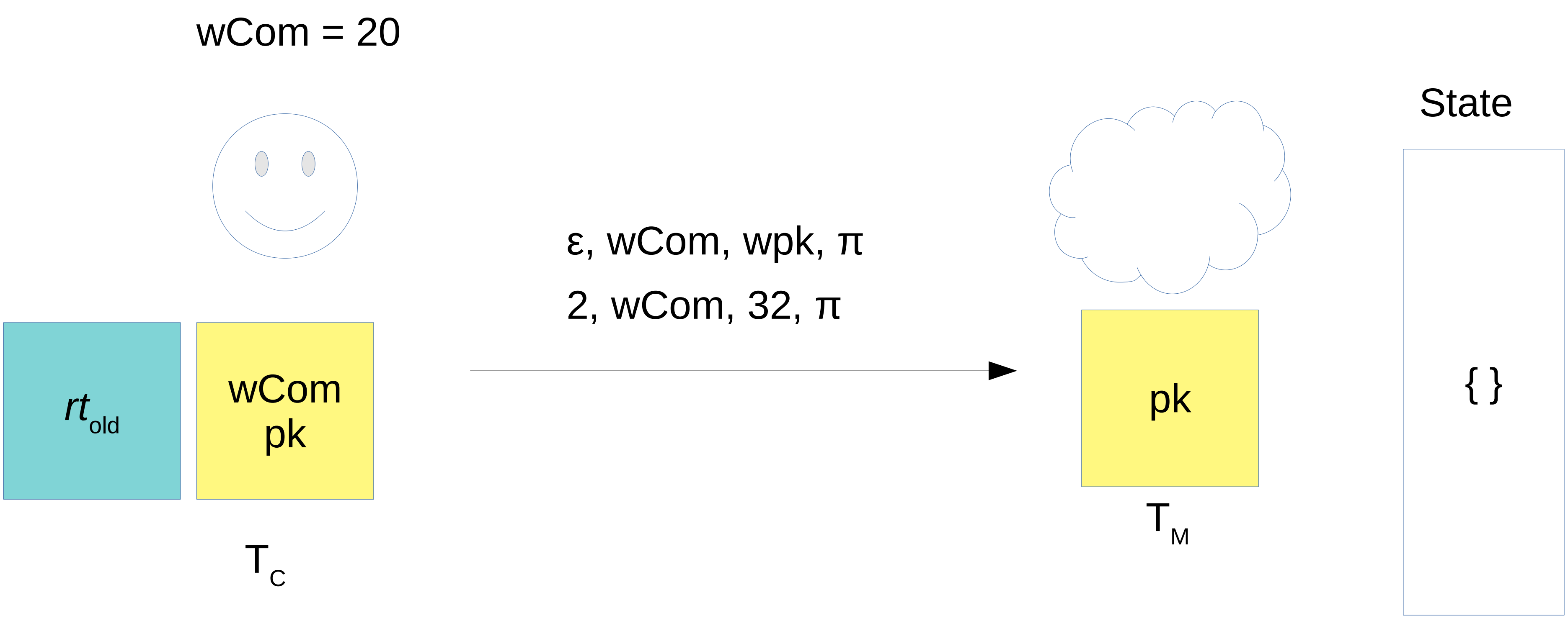


Bidirectional : Pay



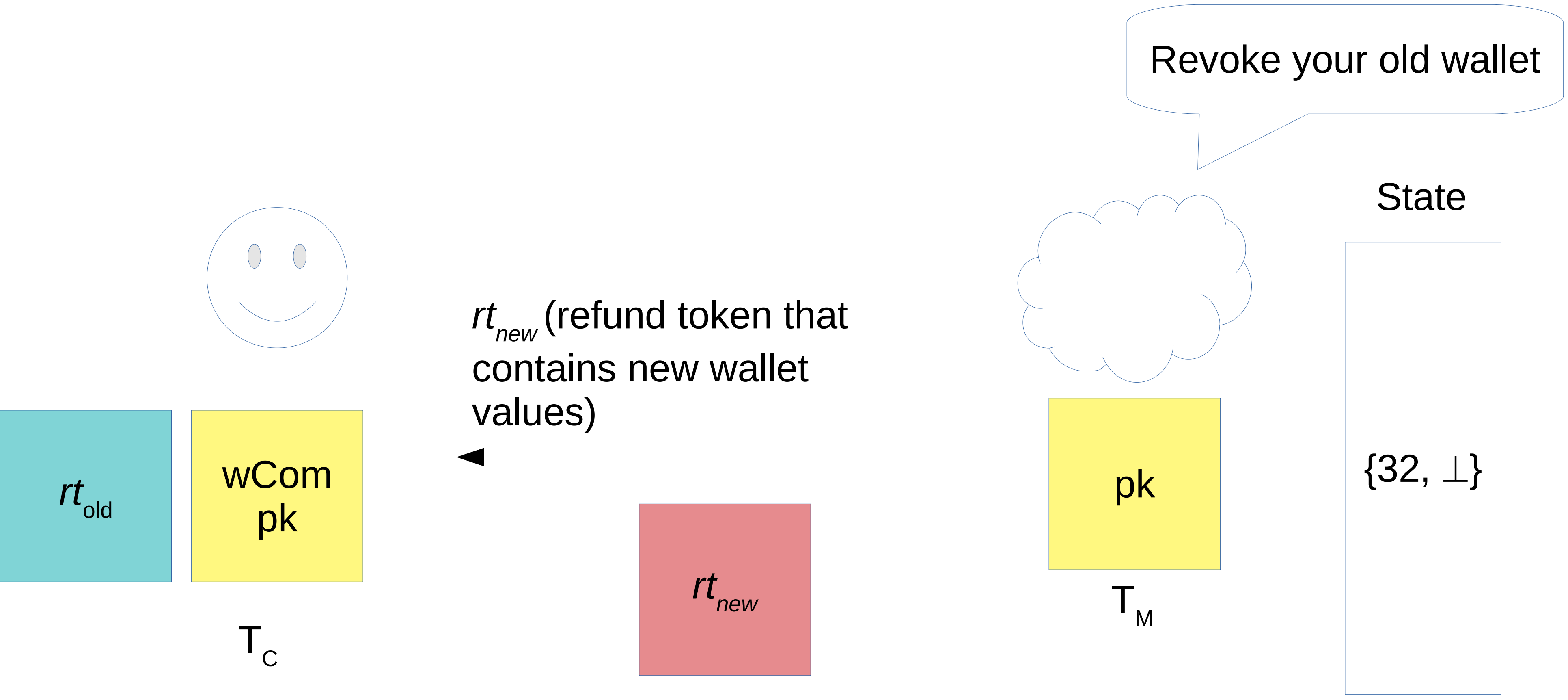
- Merchant verifies that the customer has revoked their old wallet

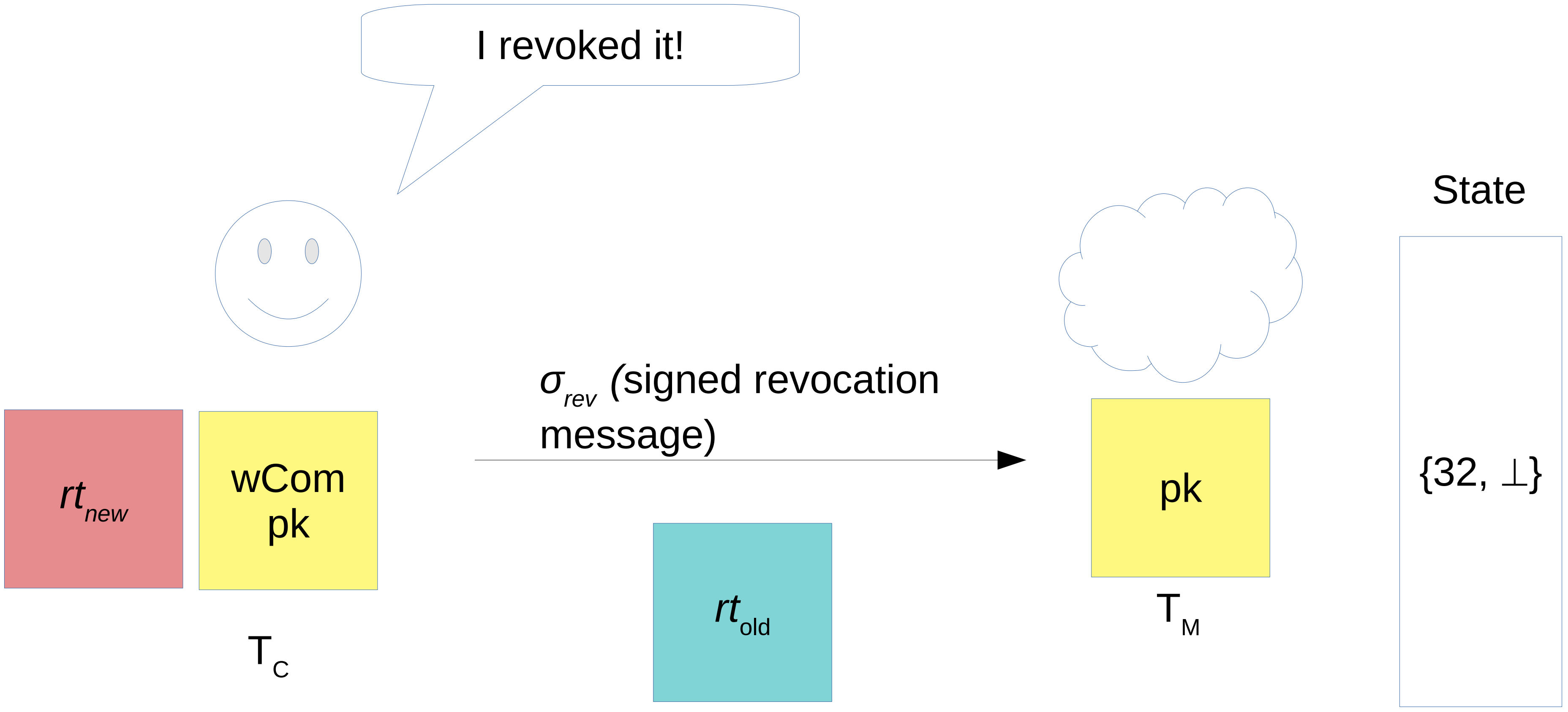
Bidirectional Pay Example



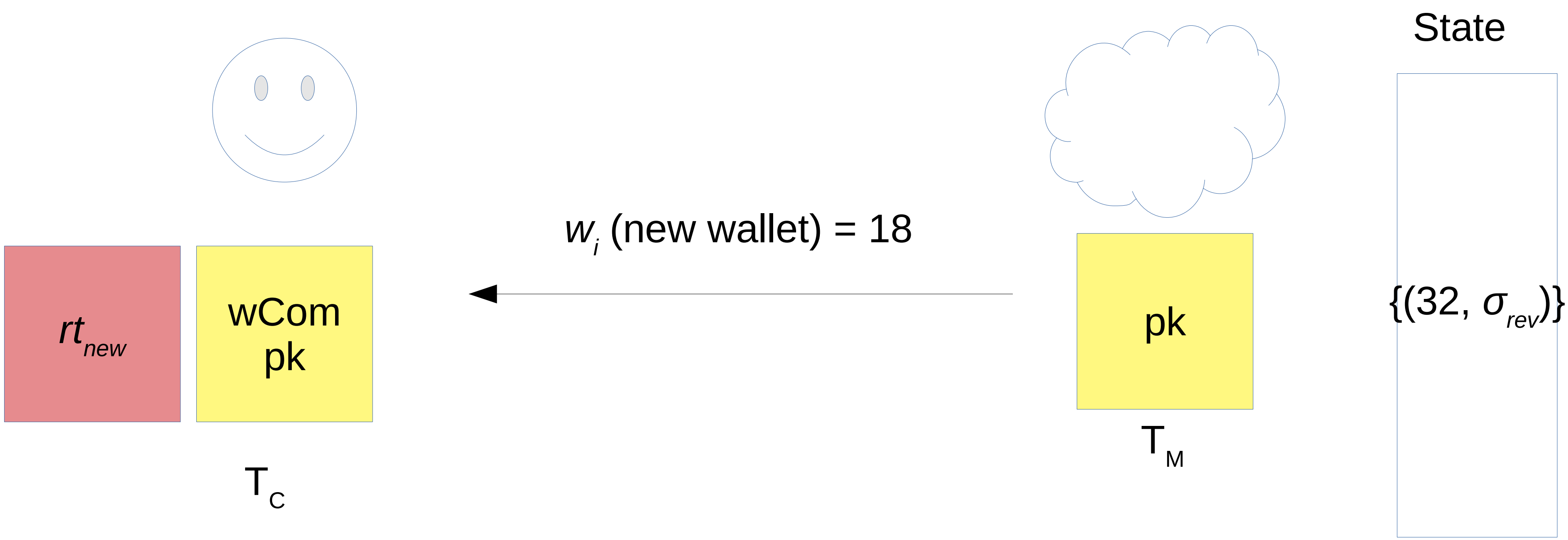
$$\pi_2 = PK\{(wpk', B, r', \sigma_w) : \ \mathsf{wCom}' = \mathsf{Commit}(wpk', B - \epsilon; r') \\ \wedge \mathsf{Verify}(pk_m, (wpk, B), \sigma_w) = 1 \\ \wedge\, 0 \leq (B - \epsilon) \leq \mathsf{val}_{\mathsf{max}} \ \}$$

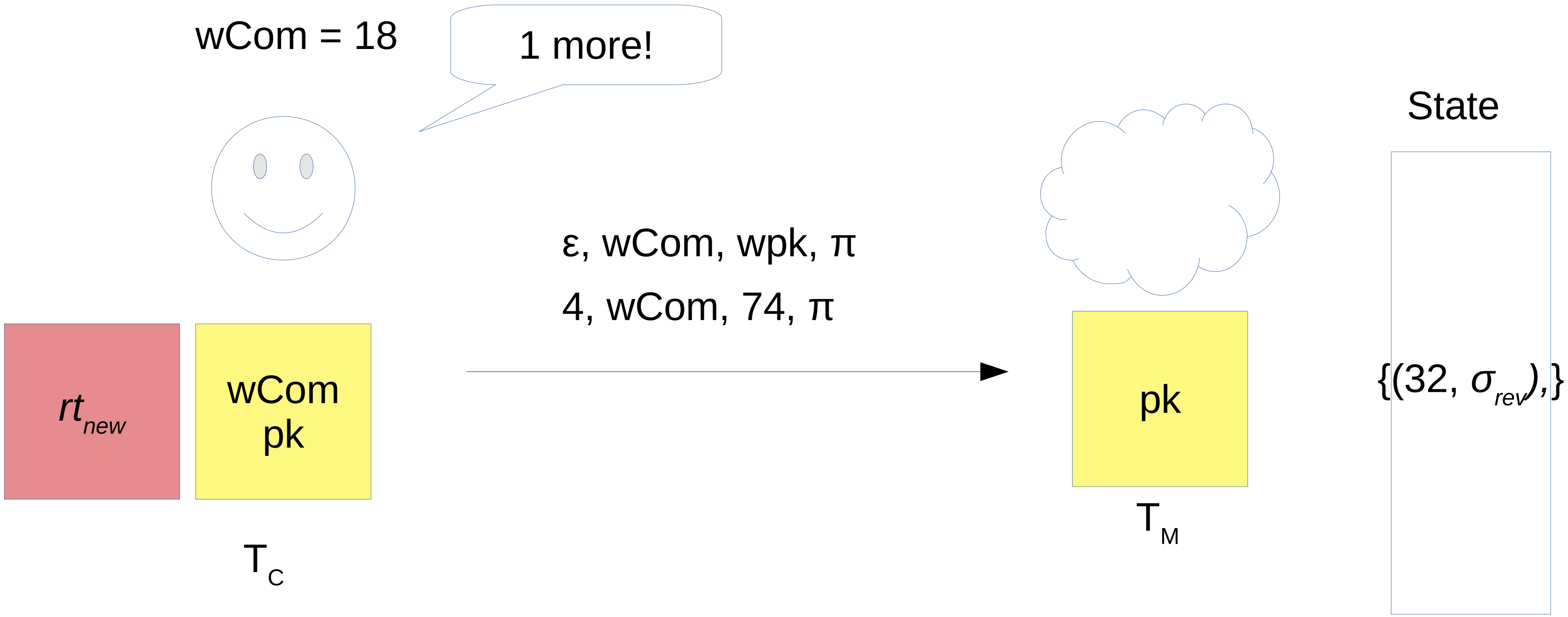
Bidirectional Pay Example



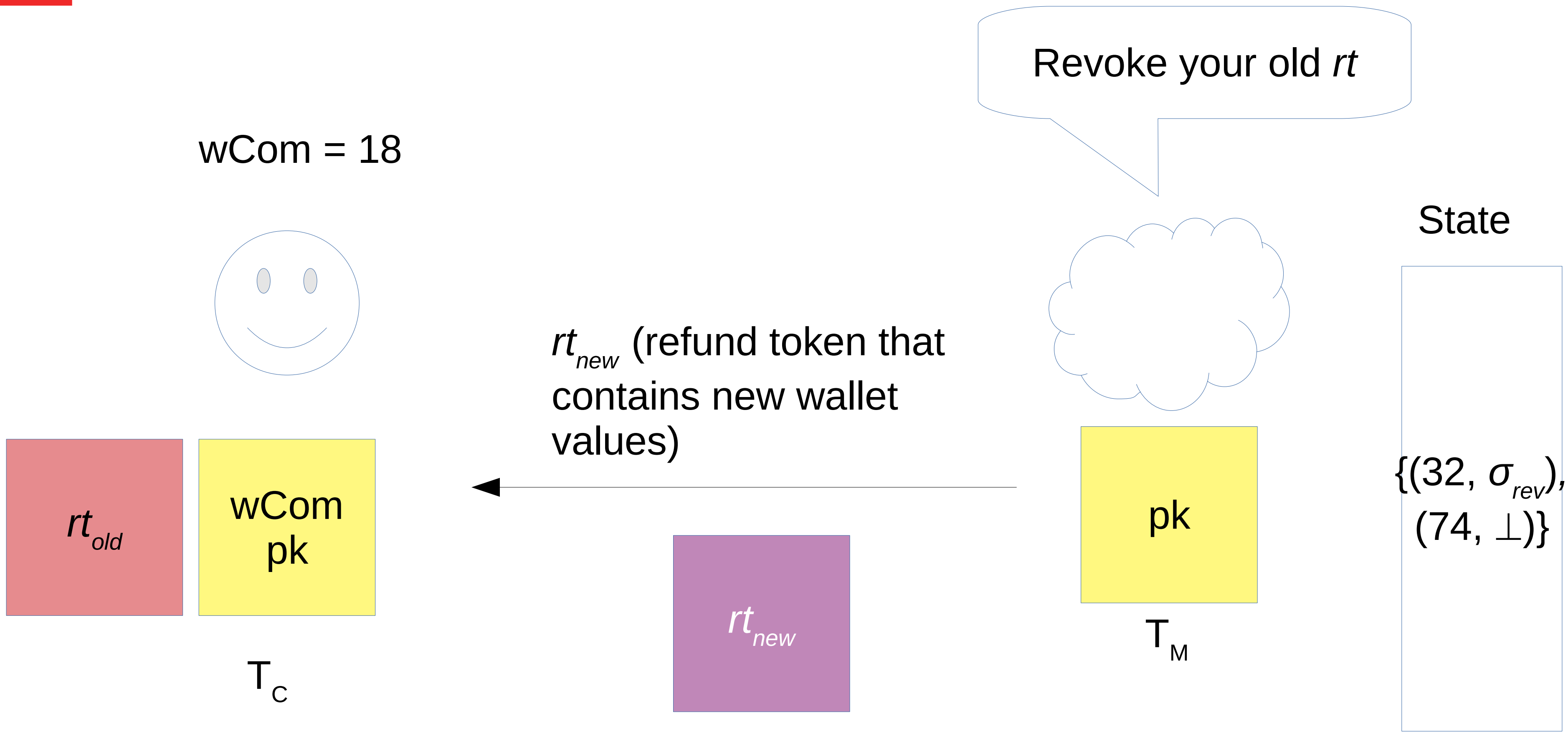

I revoked it!
State
σ_{rev} (signed revocation message)
rt_{new}
wCom pk
T_C
pk
$\{32, \perp\}$
T_M
rt_{old}

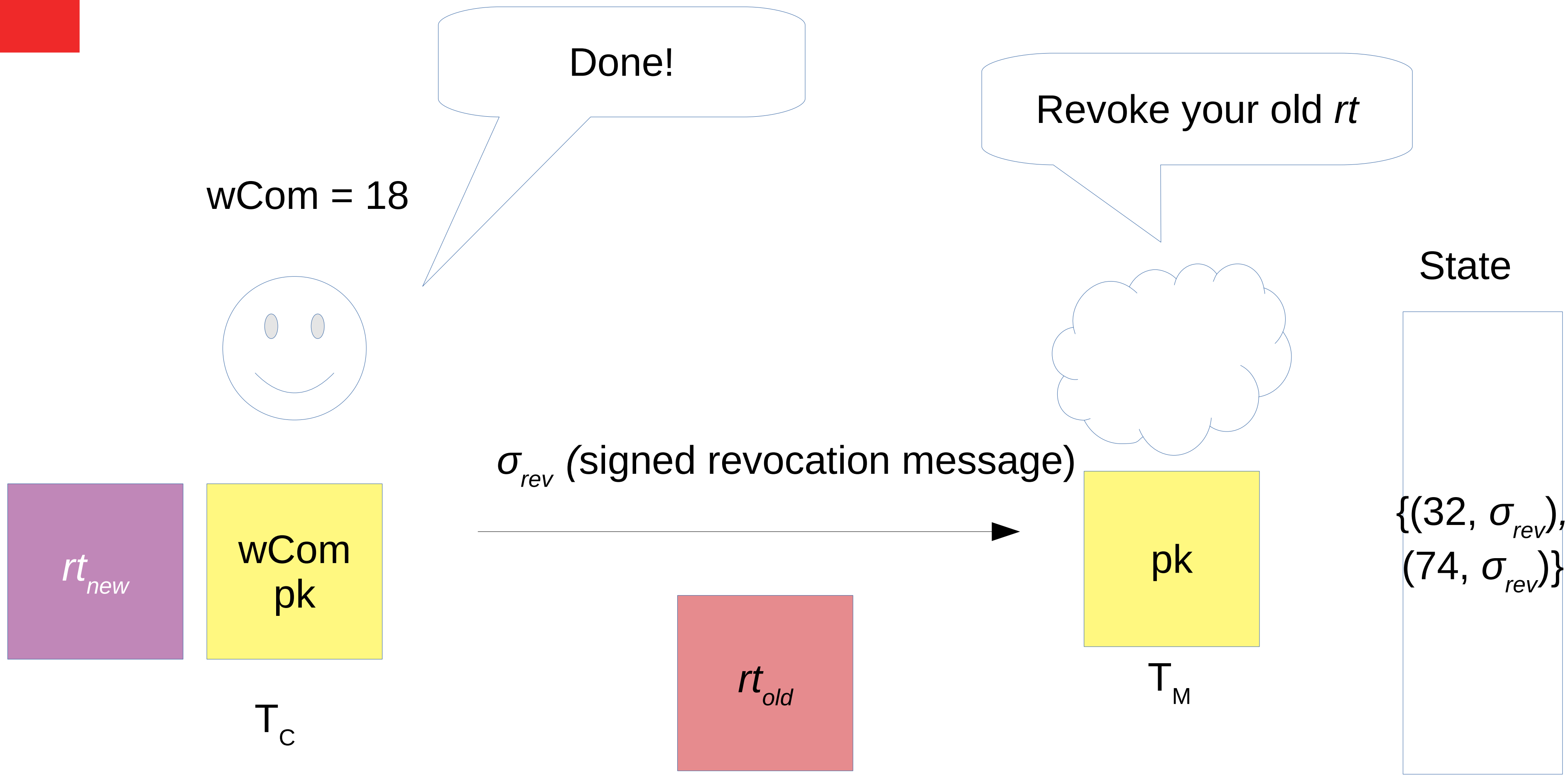
wCom = 20 → 18

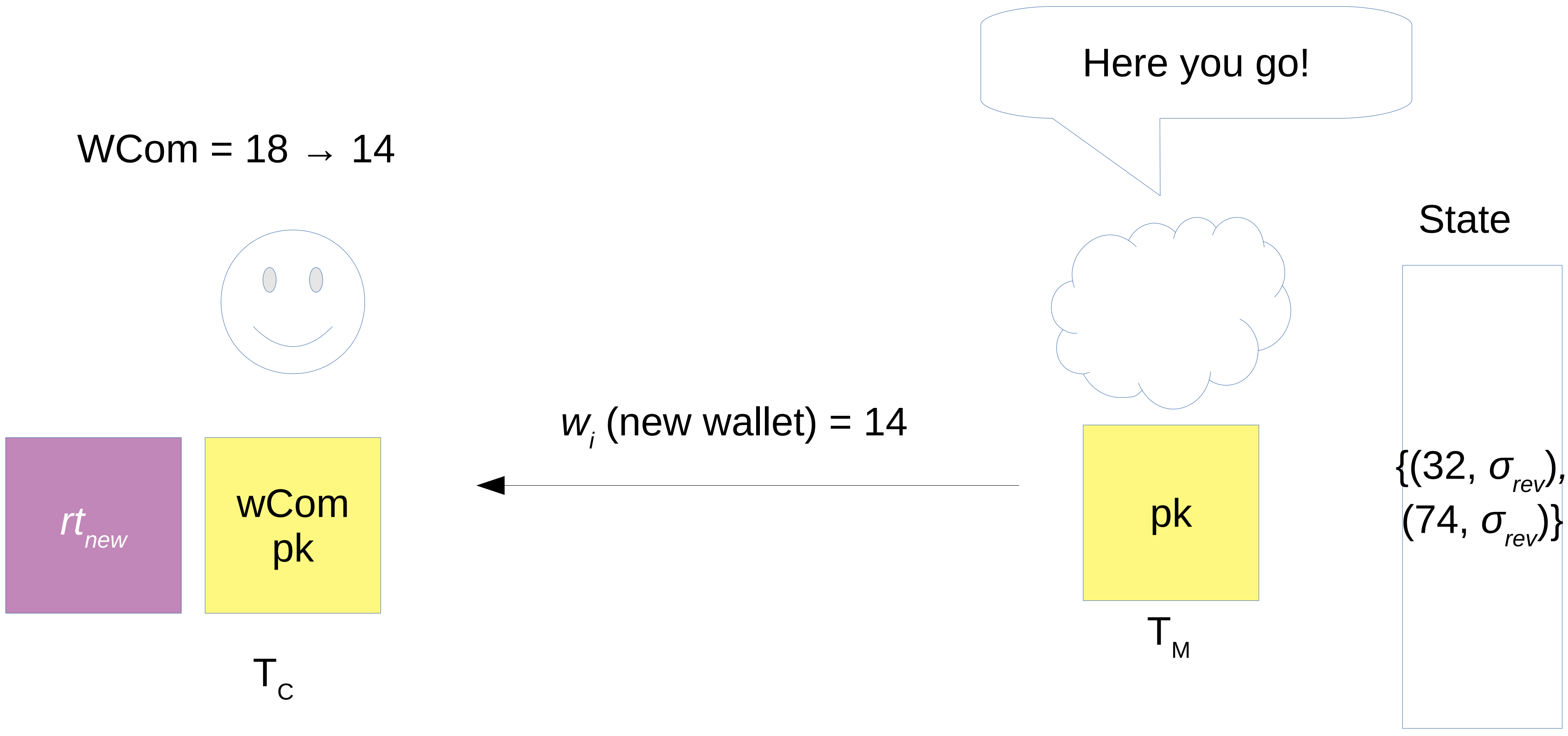




wCom = 18
1 more!
ε, wCom, wpk, π
4, wCom, 74, π
rt_{new}
wCom
pk
T_C
pk
T_M
State
{(32, σ_{rev}),}


Revoke your old *rt*
wCom = 18
State
rt_{new} (refund token that contains new wallet values)
rt_{old}
wCom
pk
T_C
rt_{new}
pk
T_M
{(32, σ_{rev}),
(74, ⊥)}


Done!
Revoke your old rt
wCom = 18
State
σrev (signed revocation message)
rtnew
wCom
pk
TC
rtold
pk
TM
{(32, σrev),
(74, σrev)}


Here you go!
WCom = 18 → 14
State
w_i (new wallet) = 14
rt_{new}
wCom
pk
pk
$\{(32, \sigma_{rev}),$
$(74, \sigma_{rev})\}$
T_C
T_M

Bidirectional : Closure

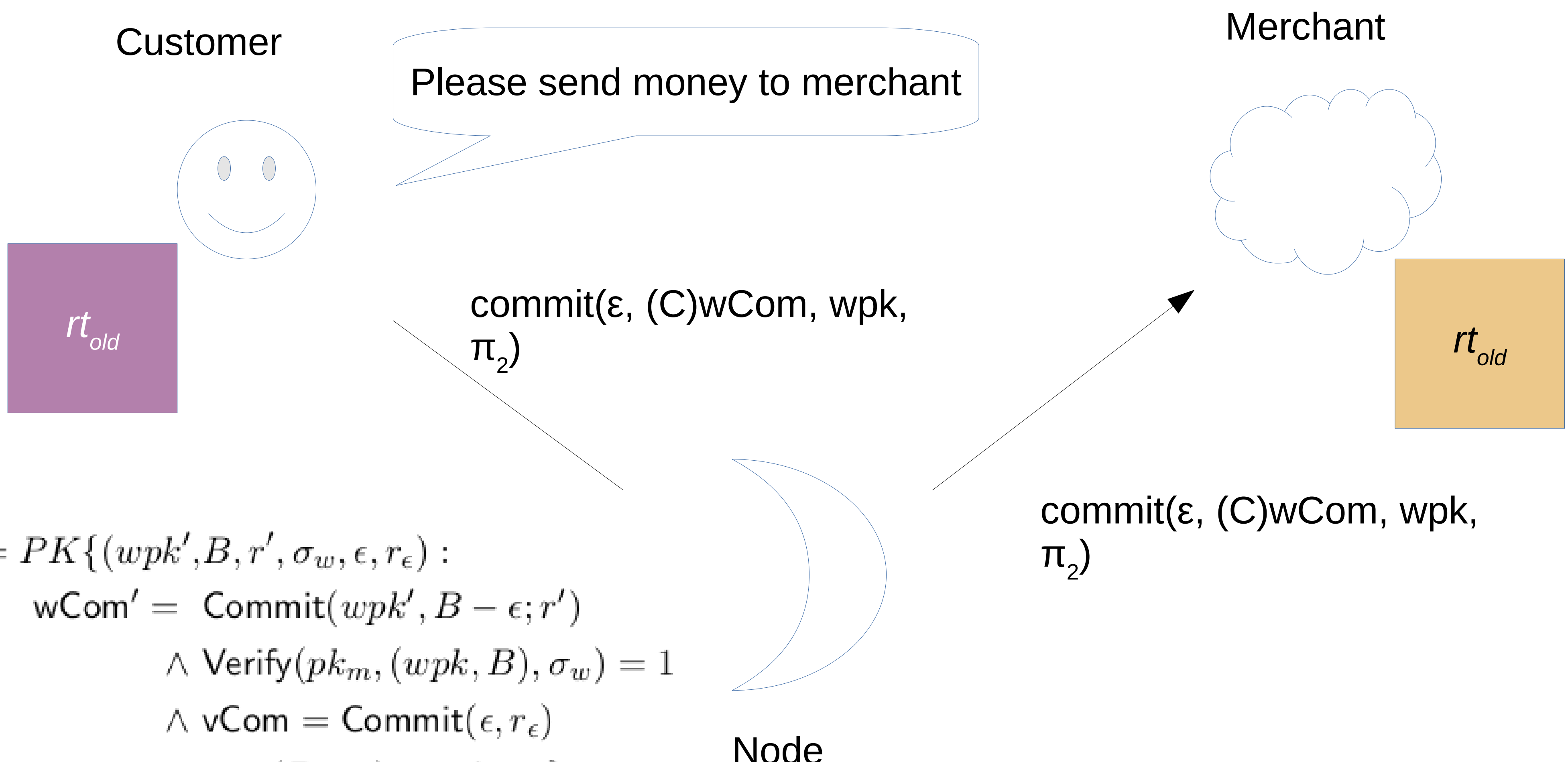
- Like Unidirectional
  - Need rc from both parties to initiate channel closure and release to network for confirmation
- This though, customer's rc_C is reliant on the *rt* given by the merchant
  - Customer signs the *rt* version given by the merchant and that gets outputted as rc_C
- Merchant takes the customer's rc_C and verifies that the *rt* in the *rc* is not an already revoked *rt*
  - If the *rt* given is already revoked, then include it in the merchant's rc_M and send it up to the network for the Resolve protocol

Bidirectional : Resolve

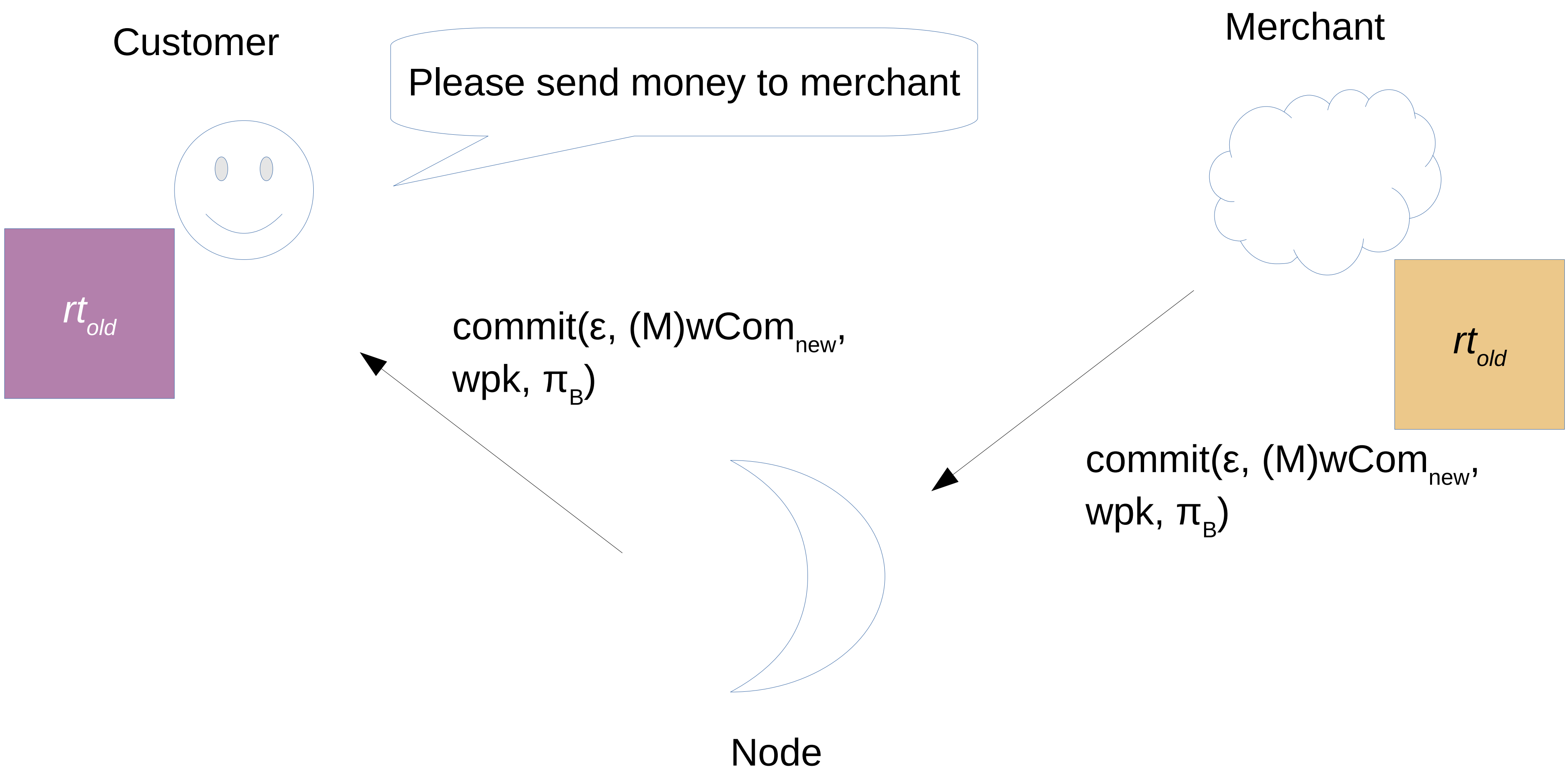
- Network runs resolve and checks if the *rc* given are either valid or not
  - If the customer's *rc* is invalid, return $\perp$ and give all the money in the channel to the merchant
  - If both parties' *rc* is invalid, reject the channel's result

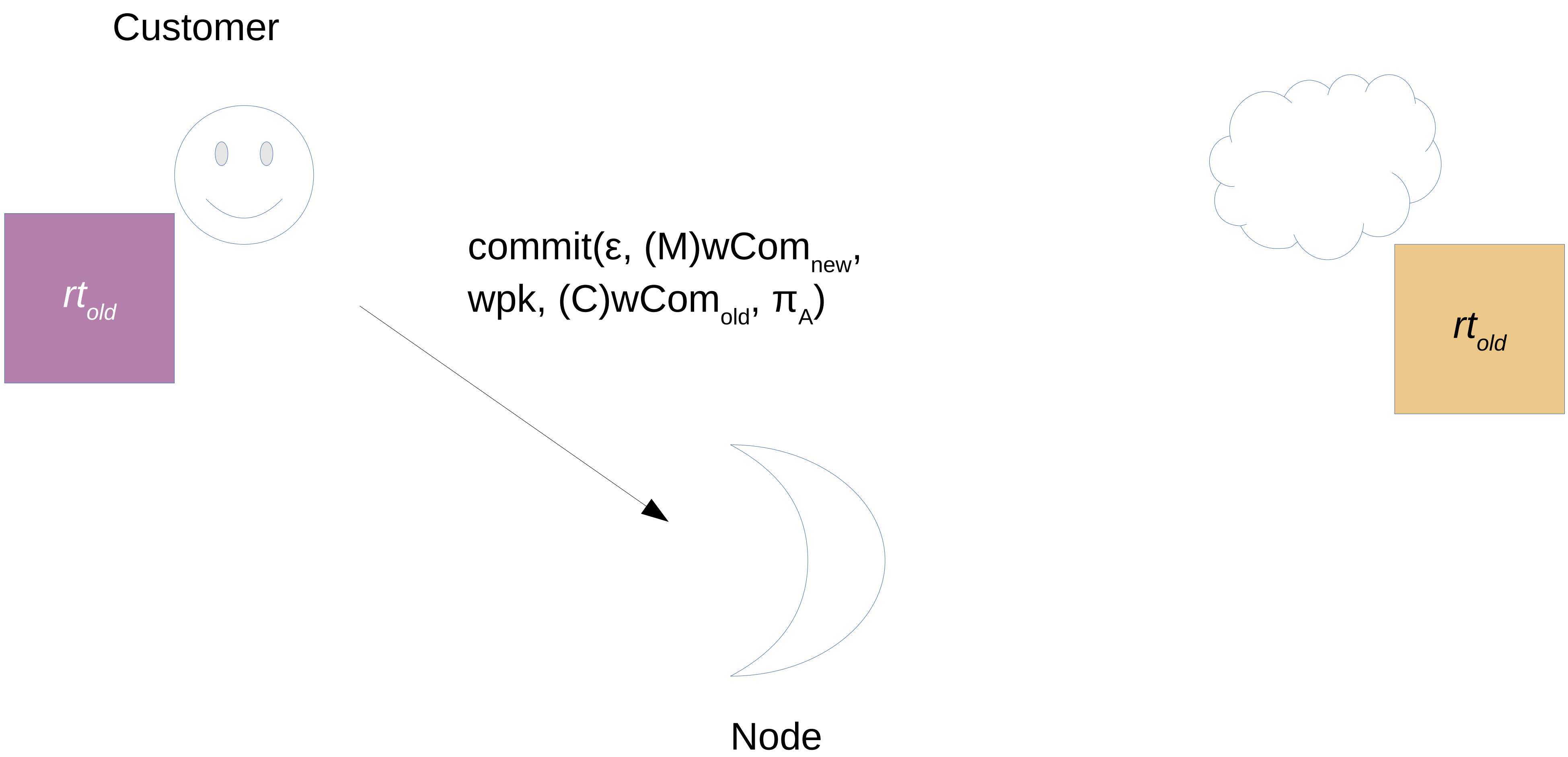
Indirect Payments

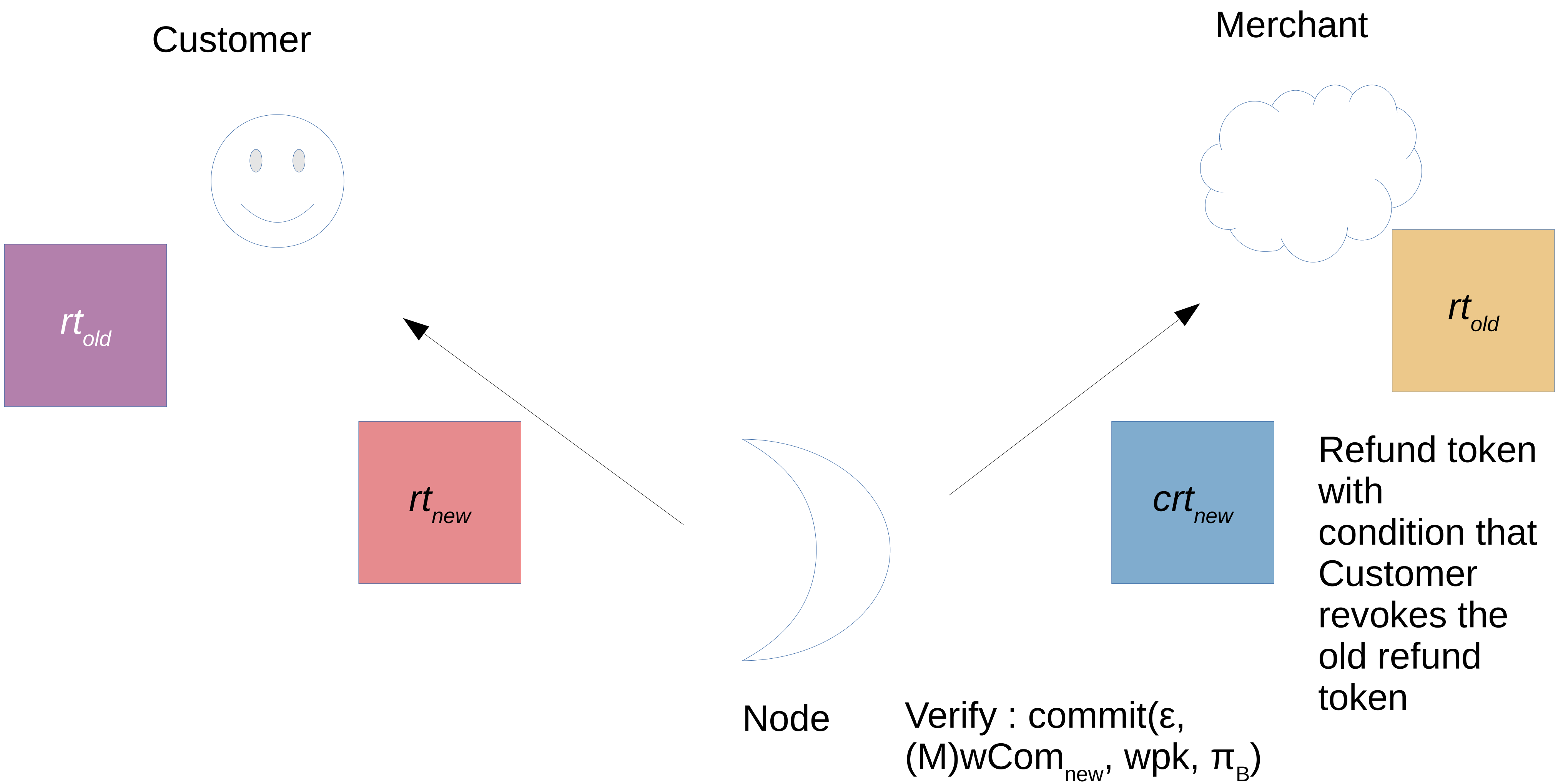
- Bidirectional third party payments
- A $\leftarrow$ $\rightarrow$ I $\leftarrow$ $\rightarrow$ B
- “No purely cryptographic solution to this problem since it’s in essence fair exchange”
- There are still ways to leak information of both parties due to abortion
- “Fundamentally difficult to avoid in an interaction protocol”
- Utilize blockchains for aborts: either processes the entire transaction from start to finish without errors or the entire transaction is voided (people get their money back)

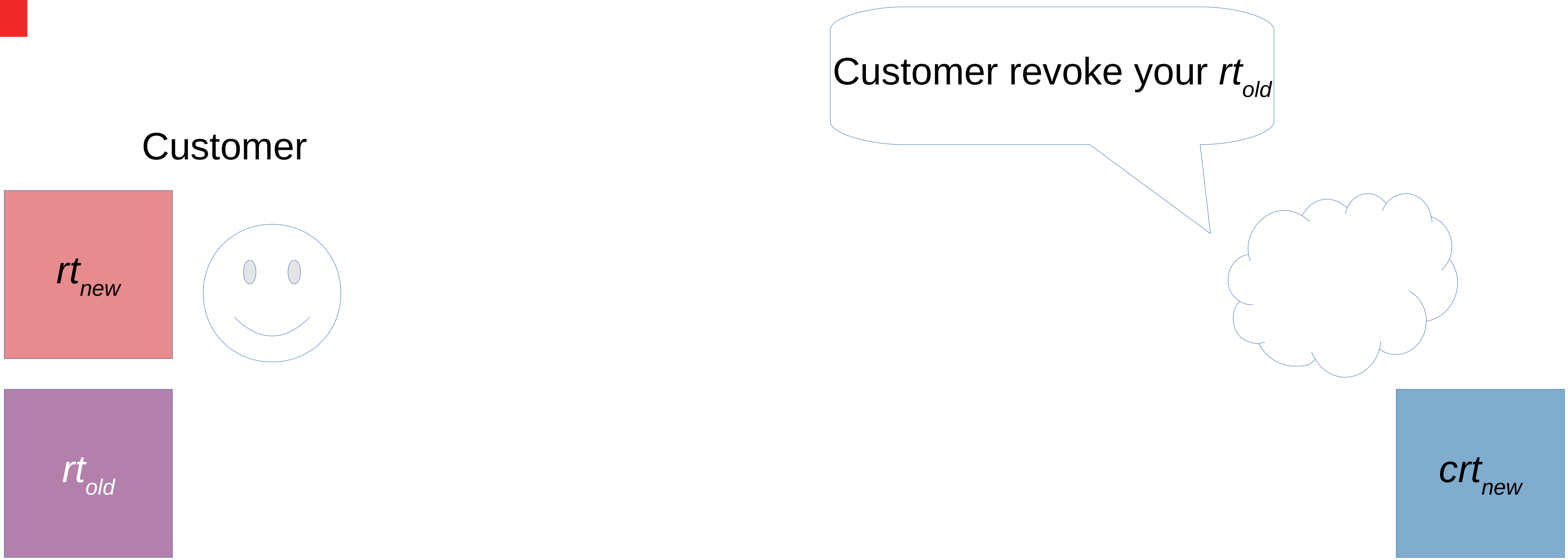


$$
\begin{aligned}
{}_2 = PK\{(wpk', B, r', \sigma_w, \epsilon, r_\epsilon) :\ & \\
\mathsf{wCom}' =\ & \mathsf{Commit}(wpk', B - \epsilon; r') \\
& \wedge \mathsf{Verify}(pk_m, (wpk, B), \sigma_w) = 1 \\
& \wedge \mathsf{vCom} = \mathsf{Commit}(\epsilon, r_\epsilon) \\
& \wedge 0 \leq (B - \epsilon) \leq \mathsf{val}_{\mathsf{max}}\ \}
\end{aligned}
$$

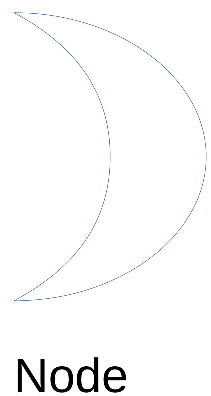

Customer
Please send money to merchant
Merchant
rt_old
commit(ε, (M)wCom_new, wpk, π_B)
rt_old
commit(ε, (M)wCom_new, wpk, π_B)
Node

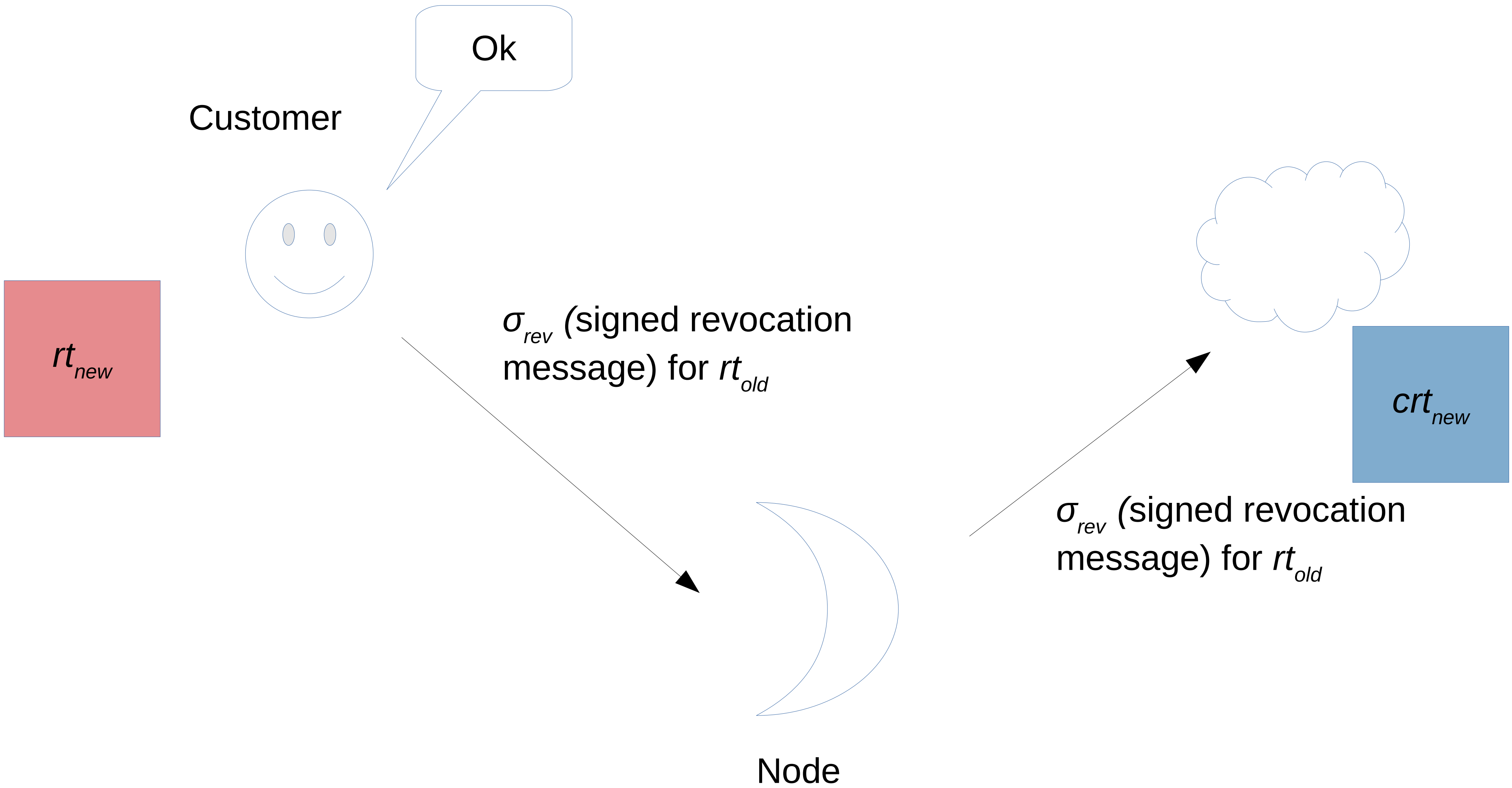

Customer
rt_{old}
commit(ε, (M)wCom$_{new}$,
wpk, (C)wCom$_{old}$, π$_A$)
rt_{old}
Node

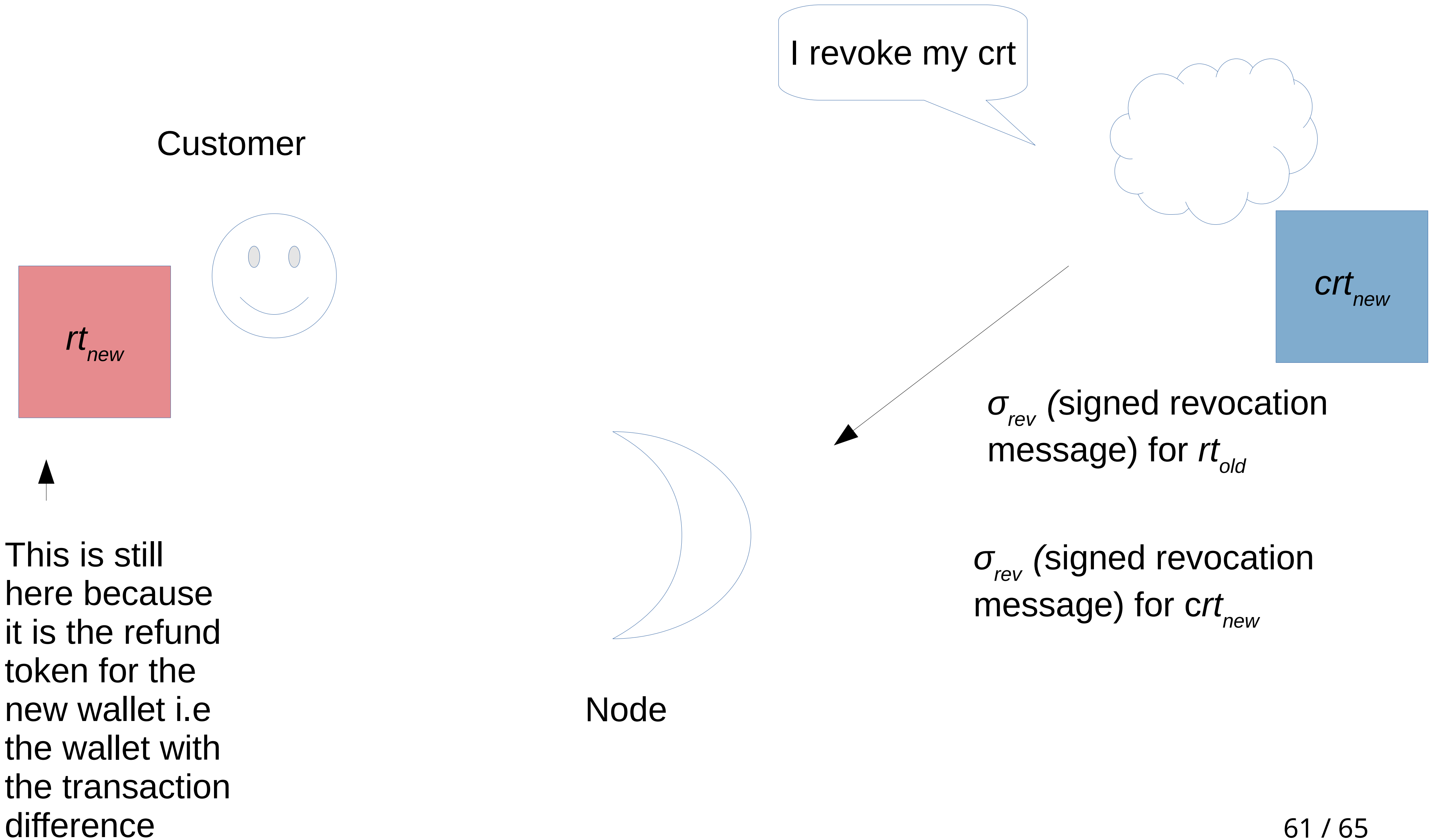

Customer
Merchant
rt_{old}
rt_{new}
crt_{new}
rt_{old}
Node
Verify : commit(ε, (M)wCom$_{new}$, wpk, π_B)
Refund token with condition that Customer revokes the old refund token

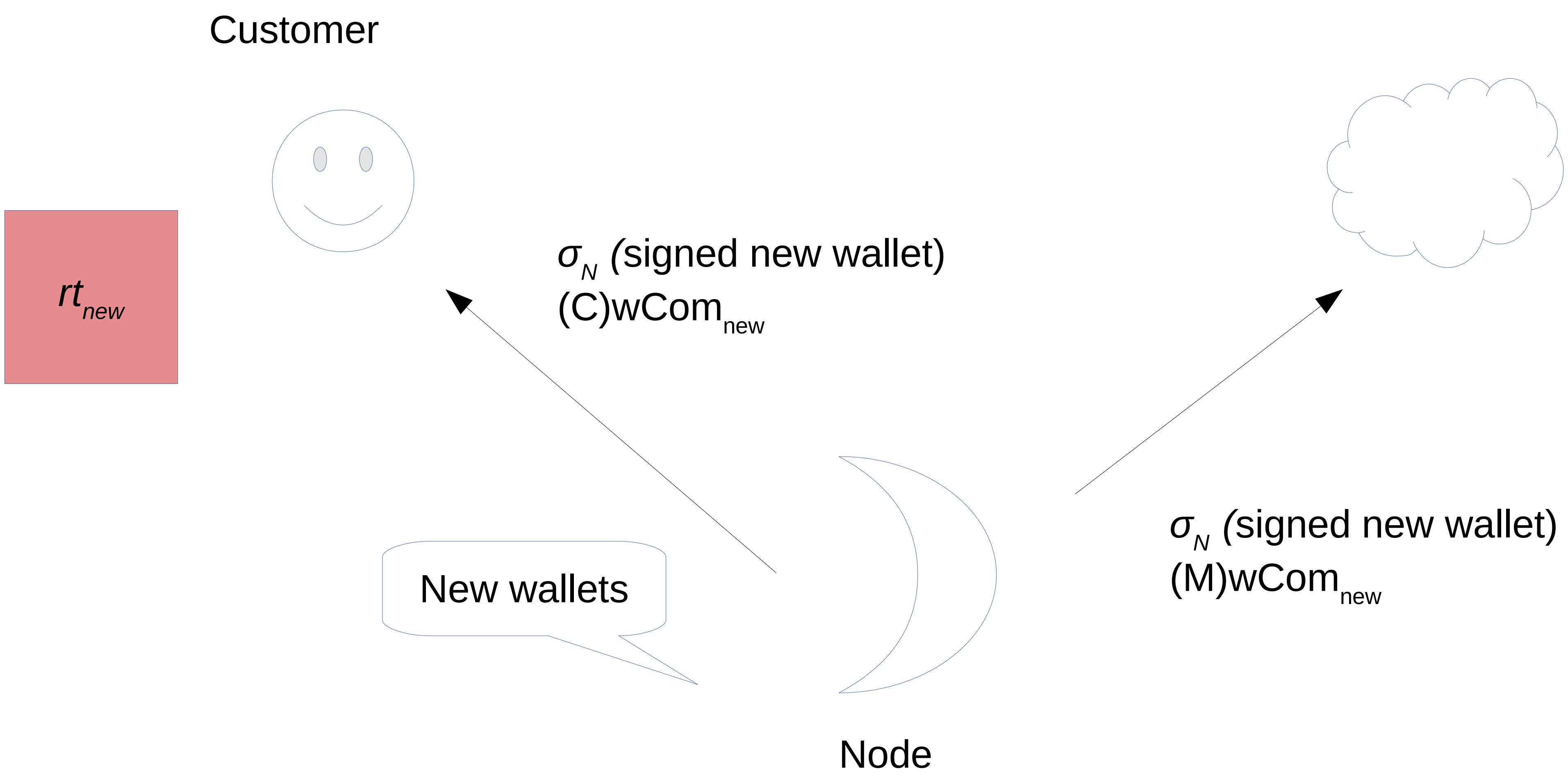


Customer has 2 options:
Either sign revocation or
cash out the current refund
token




Ok
Customer
rt_{new}
σ_{rev} (signed revocation message) for rt_{old}
crt_{new}
σ_{rev} (signed revocation message) for rt_{old}
Node


I revoke my crt
Customer
rt_{new}
crt_{new}
σ_{rev} (signed revocation message) for rt_{old}
σ_{rev} (signed revocation message) for crt_{new}
This is still here because it is the refund token for the new wallet i.e the wallet with the transaction difference
Node


Customer
rt_{new}
σ_N (signed new wallet)
(C)wCom$_{new}$
New wallets
σ_N (signed new wallet)
(M)wCom$_{new}$
Node

- The transaction now must occur between A → I → B because I has the revocation tokens
- If A tries to use a revoked refund token, the network will notice and punish A during Resolve protocol (punishment not specified)
- If A does not revoke the refund token, then the channel closes no transaction of money occurs between the parties

Conclusion

- There is a possible threat of anonymity and privacy tracking during a intermediary abort, which results in both the merchant and customer requesting refund from the network which can be linked
- This threat is limited however through the use of anonymous/temporary coins with alternative identifiers
- The malicious node can only call abort once per channel, and each channel creation results in its own unique IDs thus its mathematically difficult to link its customers and merchants
- BOLT achieves anonymity and privacy quite well

References and Sources

- https://en.bitcoin.it/wiki/Lightning_Network
- https://www.youtube.com/watch?v=qTB8ViRY9Ps -Blind signatures
- https://www.ece.cmu.edu/~ece734/fall2015/recitations/7-18734R-Commitments.pdf

  Pedersen's commitment scheme
- https://cs.nyu.edu/courses/spring12/CSCI-GA.3210-001/lect/lecture14.pdf

  Pedersen's commitment schemes
- Green, Matthew & Miers, Ian, (2016), *Bolt: Anonymous Payment Channels for Decentralized Currencies*
- *https://www.doc.ic.ac.uk/~mrh/330tutor/ch06.html* - Discrete Logs
- https://usa.visa.com/run-your-business/small-business-tools/retail.html