Secret-less secured payment system for inter-broker communications

Jean-Philippe ABEGG
ICUBE, University of Strasbourg, France

Quentin BRAMAS
ICUBE, University of Strasbourg, France

Thomas NOEL
ICUBE, University of Strasbourg, France

***Abstract*—The growing importance of the data in today's applications, such as machine learning, makes merchandising them more appealing, but it opens various challenges. Indeed, we want data to be delivered in a fast, reliable, and secure manner. This means we want guarantees about the payment and the correct delivery of the data, while still offering the ease of existing publish/subscribe protocols.**

There exists no state-of-the-art protocol that answers to all those challenges. Most of them lack the security properties needed for merchandising data and the few secured propositions do not scale well with the number of buyers, which prevents them from global use.

In this paper, we present a data payment system based on SUPRA, a publish/subscribe protocol having delivery guarantees. With our solution, it is possible to securely sell data while sharing them using the publish/subscribe model, which is known for its scalability in one-to-many communications.

After presenting how our system works, we explain how it answers the various challenges of merchandising data and compare it to the other state-of-the-art solutions.

***Index Terms*—blockchain, publish/subscribe, data payment**

I. Introduction

Having a large amount of high-quality data is crucial for many applications, in particular for services that use artificial intelligence. The main producers of data include connected devices sensing their environment (referred to as the internet of things) and online services themselves that log users' events. These data are then used to analyze, monitor, and predict events or behaviors using tools such as machine learning. The growing importance of the data has opened the possibility for it to be merchandised. Selling data raises several challenges: the price of individual data is negligible; data should be received quickly after it is produced and easy to manage; data must be encrypted and the communications auditable. In this paper, we propose a solution to connect data producers to data consumers offering strong guarantees about the delivery and the payment of each data message.

The Internet of things enables real-time updates of an environment. The owner of a set of sensors deployed in an area gathers measurements from these sensors in a server, called *gateway*. The quality and the quantity of the data depend on the sensors and the settings. Some data consumers can be interested in such data, for instance, to know the evolution of a specific environment, but do not want to deploy sensors themselves, either because they are interested only for a short period, or because they do not have access to the environment. They can be potential clients for the entity that produces the data and are willing to sell these data.

Our proposed solution is based on a publish/subscribe protocol so that managing data is as simple as with existing IoT-oriented protocols such as MQTT [2]. Moreover, data messages are transmitted directly between the data-producer server and the data-consumer server. Also, we offer payment guarantees for the vendor and delivery guarantees for the buyer. For the vendor, we prove that it will receive the money according to the amount of data sent to the buyer. For the buyer, we prove that it will receive the data correctly and it will be able to detect any missing messages.

Blockchain and cryptocurrencies can be used for creating this kind of selling protocol. Indeed, crypto-currencies have a smaller granularity than classic currencies. This means that it is possible to buy something for less than 0.01$ (or £, €, ...) if we put aside the transaction fees. So, with the help of the blockchain, we can put a price on each sensor data individually. In contrast, with a classic currency, we have to round the price if it is not a multiple of the currency's smallest denomination.

On top of this, blockchain can add the security properties needed for this kind of protocol to be used safely by users, because transactions are visible for all blockchain users, which means that anyone can verify the data payments, or let an automated application like a smart-contract do it.

There are several propositions for data payment protocol with blockchain [4], [10], [11] but they do not take into account malicious behaviors from the users. This means that the buyer is not sure to get its data, and the vendor is not sure to get its payment. This issue prevents the usage of these protocols in real-life scenarios. Also, these propositions do not take into consideration important properties of the blockchain such as the high confirmation time of transactions and the expensive fees per transaction. On the other hand, Lightning Network [9] is a proposition taking into account all these needs but is not adapted for publish/subscribe data-centric communication, a model used to share IoT data.

Contributions: In this paper, we present an extension of SUPRA [1], a publish/subscribe protocol, that allows brokers to buy/sell data from other brokers. This extension offers payment and delivery guarantees and avoids as much as possible the use of the blockchain, to reduce the transaction fees and increase the data delivery rate. We prove the security of our

ISBN 978-3-903176-49-2

protocol, in particular, the mechanisms preventing malicious users to steal data without paying. Finally, we compare our proposition to other state-of-the-art solutions and show under which conditions which solution is more interesting.

II. Related work

A. Blockchain and crypto-currencies

Blockchain is a distributed ledger technology that was first presented by Satoshi Nakamoto in 2008 [7]. The purpose of this technology is to make a network of nodes maintaining an immutable distributed ledger of transactions. These transactions are grouped into blocks. Each block is linked to the previous block using hash pointers and then added to the ledger. It creates a chain of blocks, hence the name blockchain.

The author of a transaction is identified with a pair of public/private cryptographic keys. Each transaction is signed by its author using the private key and blockchain nodes verify the signature using the author's public key. The transaction is sent to a node of the blockchain and then broadcast over the network. Each node saves incoming transactions in a pool used to build the next blocks. Once a new block is validated, transactions integrated into this block are removed from the pool. The way blocks are appended is a result of a consensus algorithm that depends on the blockchain technology (*e.g.*, in Bitcoin a single node is elected to append the next block).

Once a block is added to the chain, the transactions in it cannot be modified (or at least the probability that a modification can be made decreases exponentially fast over time). That is why data on the blockchain is considered immutable. This property remains true as long as a certain amount of nodes follow the protocol honestly. The minimum amount of honest nodes depends on the consensus algorithm and the blockchain implementation.

Crypto-currencies are the most known applications of blockchain technologies. In this type of application, the transactions in the ledger contain a transfer of tokens from one user to another. The purpose is to create a payment system that does not rely on a central authority to run, like banks from classic currencies, but where every user can trust the accounts' balance with the help of the blockchain transparency. The most famous crypto-currencies is Bitcoin, which is also the first application using blockchain.

Aside from the decentralized property, crypto-currencies have an interesting payment property. They allow a smaller granularity in payment than classic currencies. For instance, in Bitcoin, the main token is also named Bitcoin but it is not the smallest transferable token. Indeed, the smallest denomination of a Bitcoin is called a Satoshi which is 10^{-8} Bitcoin, and based on exchange platforms, 1 Satoshi is less than 0.01 €.[1] Which means that with Bitcoin, we can buy/sell individually things that cost less than 0.01€. This property is also present in other crypto-currency, for instance, in Ethereum [3] the smallest denomination of an Ether is called the Wei, which is 10^{-18} Ether, and also has a value smaller than 0.01€.

[1]As of July 2022, 1 Satoshi = 0.00021 €

This small denomination in crypto-currencies is an interesting property for sensor data because each data individually has a negligible monetary value, so it could be a problem with classical currency if you have to sell data that is worth less than 0.01€. Whereas with crypto-currencies, you can directly sell each data individually, if we assume that the price for a data is equal to, or higher, than the monetary value of the smallest denomination of the token. Unfortunately, two issues prevent users to sell sensor data with crypto-currencies: the delay to add a transaction in the ledger, and the transaction fees. Each transaction contains fees. Those fees are used to pay the network for its work. If the fees are higher than the price of a single data, it is uninteresting to use crypto-currencies, because users will spend more on fees than on data. Also, adding a transaction takes time, for instance in Bitcoin, once a transaction is in a block, since the chain can fork, one usually waits one hour before being sure that the transaction is in the main branch of the chain and will stay in the ledger. This can prevent the usage of crypto-currencies for real-time data, if the vendor waits for the payment before sharing data, the buyer will have to wait one hour for each data before receiving it. The Lightning Network [9] resolves both issues, but we will explain in the next section why it is not a good solution for IoT oriented environment.

B. Data payment protocol

The goal of a data payment protocol is to ensure that the buyer receives the data and that the vendor receives the payment. To do so, the protocol have to execute an atomic swap [5] between the data and the payment. An atomic swap is an exchange between two or more users where, if all users are honest, the exchange is made correctly, but if one user if malicious and try to deviate from the protocol, then all honest users cannot end-up in a situation worse than the one before executing the swap.

To execute an atomic swap with a blockchain, we can use the Lightning Network (LN) [9]. It is a decentralized payment protocol that allows two users to create a lightning channel. This channel allows users to exchange as many transactions as they want instantly without paying fees. Let's assume that A and B want to frequently exchange tokens from A to B or from B to A. To set up the lightning channel, A and B lock some funds with a transaction signed by both users called a commitment transaction. This transaction is added to the ledger. Once it is done, A and B can exchange off-chain an arbitrary number of transactions. Every time A and B want to exchange tokens, they both create a revocable commitment transaction with their current agreed balances. If any of them wants to leave with the money, it will get the last agreed balance. Each new revocable commitment transaction revokes the previous one, hence, no one can use the previous commitment transactions to leave the network with the previously agreed balance (which is not valid anymore). Indeed, if one user is malicious and tries to do so, the other user can prove it and receive the funds locked by both user in the commitment transaction as a reward.

One could extend Lightning channels to allow a data transfer, hence ensuring that a commitment transaction takes effect only when data is correctly transferred. However, this would add complexity to a protocol that already require complex secret exchange between participants. Moreover, LN would require each broker to keep all the revoked transactions. Our solution achieve the same goal with a much simpler approach, using the fact that data transfer is directed from the seller to the buyer, which removes the necessity to store secret-based revocable transactions.

In the literature, we find several data payment protocols. For instance, in [11], the authors present a system of payment for publish/subscribe protocol called PPSP. In this protocol, the publisher delegates the payment handling functionality to the broker. The publisher sends data and the data price to the broker. Then the broker will charge the subscribers and forward the crypto-currencies to the publisher's wallet. The issue of this proposition is that the payment verification adds delay before delivering each data. Indeed, before forwarding the data to the subscriber, the broker waits for the payment confirmation in the ledger and, as said earlier, this operation can takes several minutes or hours. This delay can prevent the usage of this proposition for real time data, where it is important for the subscriber to receive the data as fast as possible.

Another proposition is Streaming Data Payment Protocol (SDPP) [10]. This protocol uses a client/server model where the client uses a four-way handshake to order data from the seller. In this handshake, the client and the seller set which type of data will be sold, the amount D of data, the price P, and the window size K.

After the handshake, data are sent in windows of size K. It is only at the end of a window that the seller sends an invoice to the buyer to pay for the data sent in that window. When the buyer is paid, by sending the crypto-money to the seller's wallet, then the next window can start.

Sending data in window reduces delay introduces by the payment verification process present in PPSP, but it creates a new issue. Indeed, a malicious user can leave the system after receiving the window and not pay the vendor. To reduce the impact of such behaviors, the authors encourage the vendor to choose a small value for window size K, but it will increase the impact of the payment process.

D. Chen et al. [4] adapts the idea of SDPP in QoS management environments. In this version, the buyer pays before receiving the service. This time, nothing prevents the vendor to leave the system without sending data. Also, R. Nakada et al. [6] proposed an implementation of SDPP on a Raspberry Pi, but it keeps the payment issue of the protocol with the data window.

To the best of our knowledge, there is no data payment protocol for publish/subscribe environment that ensure the vendor to get the payment and the buyer to receive data. Our purpose is to propose one such protocol and to do so, we will update SUPRA [1], a publish/subscribe protocol having data delivery guarantees.

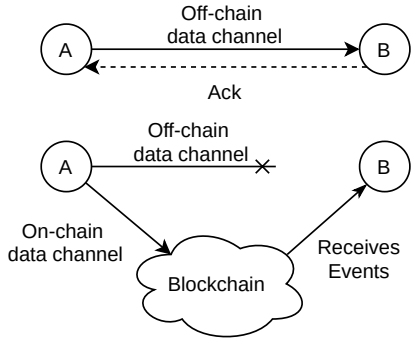

Fig. 1: The two sub-channels inside the SUPRA channel

C. SUPRA

SUPRA is a decentralized publish/subscribe protocol using blockchain. The protocol allows publishers and subscribers connected to different brokers to securely share data with delivery guarantees.

In the publish/subscribe model, the publisher and the subscriber have to trust the broker to which they are directly connected, because all the messages that they will receive, or send, will go trough this broker. If they are connected to the same broker, there is no trust issue in the system, but if they are connected to two different brokers and if the two brokers does not trust each other, how can we create a secure and reliable link between these two brokers ? Without additional assumption, a broker cannot be sure that the other broker receives the message and the broker who receives a message cannot know if some messages are missing. This issue prevents the usage of publish/subscribe with multiple brokers for use cases where missing a message is not tolerated, for instance when users share sensitive data.

The purpose of SUPRA is to secure the link between the brokers and, to do so, the protocol establishes an hybrid channel between the two brokers, like in Figure 1. This channel is split into an off-chain channel (*i.e.*a direct connection between both entities over an unreliable link), and an on-chain channel (*i.e.*messages are added in the ledger, then the destination gets the message once it is in a block). For the rest of this section, A denotes the broker that the publisher trusts and to which it is connected, and B denotes the broker that the subscriber trusts and to which it is connected.

To work, SUPRA assumes that the brokers are reliably connected to the blockchain and can receive events from the blockchain, which can be done by being a full node or by being reliably connected to a trusted full node. This also means that the brokers are not constrained devices.

To secure the link between the brokers, the first mission of SUPRA is for A to obtain a proof that a message is delivered before a delay $T_{acknowledged}$ from the first sending. Since adding information in the ledger cost money, the publisher's broker first tries to use the off-chain channel and wait for the subscriber's acknowledgment. The broker can retransmit the message as much as it wants on the off-chain channel. If

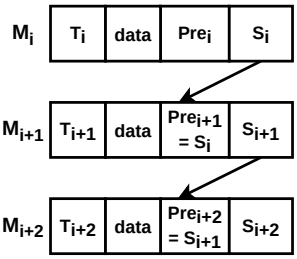

Fig. 2: Three messages chained together by signatures.

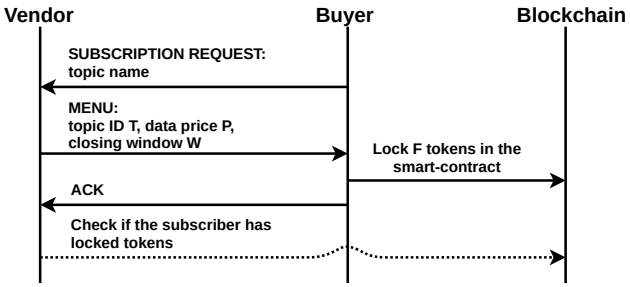

Fig. 3: The publisher and the subscriber open a channel

the publisher does not receive the acknowledgment in time, it sends the message over the on-chain channel, to be sure that the message is added in a block before $T_{acknowledged}$.

Just like in Herlihy et al. [5], which formalized the atomic swap problem, SUPRA assumes that the time to add a transaction in the ledger is bounded by a value $\Delta_{on-chain}$. Which means that A has to send the messages on-chain after a delay $T_{off-ack} = T_{acknowledged} - \Delta_{on-chain}$ from the first transmission of the message to deliver the message in time to B. No matter the subchannel used, the sender A has proof that the message was delivered in time by B: either it has the acknowledgment from B or the message is publicly available in the ledger in a block created before $T_{acknowledged}$.

To make sure that only B (and other brokers subscribed to the same topic) can read the message, the payload is symmetrically encrypted. Indeed, in [1], the authors explain how to share the symmetrical encryption key between all the interested brokers, and how to update it when a broker stops its channel with A.

On top of this delivery property for the sender, SUPRA also offers delivery property for the receiver. First, the messages are signed and timestamped by the sender, and each message contains the signature of the previous message, chaining them together just like the blocks of a blockchain with hash values. This can be seen in Figure 2. In more details, each message M_i has a field Pre_i that contains the signature S_{i-1} of the previous message M_{i-1}. It allows the receiver to detect missing messages. If B receives a message that does not contain the signature of the previous message, it knows that at least one message is missing. To retrieve missing messages, the receiver just has to wait. Indeed, since the sender has to ensure the message delivery before $T_{acknowledged}$, if the receiver waits for $T_{acknowledged}$, it will retrieve the missing messages either by receiving it directly (off-chain channel) or by receiving it in the blockchain (on-chain channel).

If the missing message are not received by B in time, it can use a smart contract, called the judge, to prove that A did not follow the protocol correctly. B presents the messages used to detect the issue and A will be unable to present a proof of delivery for the missing message (the missing message is not acknowledged nor is present in the blockchain). However, A can defend itself against false accusations. If the message is delivered correctly — it is either present in the ledger or the publisher has an acknowledgment for this message — A can submit the acknowledgment to the judge to prove that the accusation is incorrect. Since acknowledgments are cumulative (due to the signature chaining), an acknowledgment for a later message is also accepted.

To set up this communication channel, B starts a three-way handshake with A in which B indicates the topic T used for the channel. If the subscribers connected to B are interested in several topics from A, B has to open one channel for each topic but if several subscribers connected to B are interested in the same topic from A, B just have to open one channel with A.

With SUPRA, an honest user can always prove that all the messages were received in time, or not, by the correct user. This delivery property represents half the property needed for a data payment protocol, if we can add a secure payment system to SUPRA, we can have a data payment protocol capable of sharing data in a publish/subscribe manner. We will explain in the next section how to add such system.

III. Secret-less secured payment system

In this section, we explain how to add a secret-less secured payment system to SUPRA. We use the same notation and assumptions as in SUPRA. Namely, A and B denote two brokers that are connected by an unreliable link, but they are both reliably connected to the same blockchain. At least one publisher for the topic T is connected to A and at least one subscriber for this topic is connected to B.

A. Setup the communication

To start the communication, we modify the handshake between A and B used to open a SUPRA channel. The new handshake is represented in Figure 3. Just like in SUPRA, we assume that A and B both know which blockchain and which smart contract is used during the communication.

To add a payment system, we add two new information in A's answer in the handshake : the data price P, and the closing window W.

P can contain several prices based on the data's timestamp. For instance, if the publisher sells cars GPS locations, data generated just before and after business hours can cost more than data generated at night, because there are more cars on the road. The price can be also a constant value and, for the rest of the paper, we will assume that each data have a constant price P.

The closing window defines how many data can be in transit simultaneously, *i.e.*, can be sent without being acknowledged. This is important because, if the buyer B wants to close the subscription, some data could still be in transit and B is required to pay for at most W messages after the subscription is closed. With a large closing window W, A can send data at a high rate because fewer acknowledgments are required (at least one every W message). Having W data not acknowledged is not a problem because the vendor is allowed to claim the tokens to be paid for them. Of course, when W data are not acknowledged, the vendor has to wait for an acknowledgment, retransmit some data, or send them in the blockchain, following the hybrid channel specification. The value W is also a protection for the buyer because it knows that, if it fails, the vendor can ask to be paid for at most W unacknowledged messages, and not steal the whole money with dummy data. This protection is explained in more detail in section IV

The value W depends on the application. For instance, if data are shared in bursts, W can be equal to the maximum size of a burst.

B can refuse or accept the values P and W proposed in the handshake. If it accepts these values, as illustrated in Figure 3, B has to lock some funds in the smart-contract. To do so, we modify the SUPRA's judge smart contract with a function to store and manage funds in the contract. To call this function, B sends F tokens to the contract and indicates the ID of A and the topic ID T of the SUPRA channel. Locking funds with this function can be done later as well, without limitation. The details about how funds can be claimed from the smart contract are explained later.

Once the tokens are stored in the contract, the buyer B sends an acknowledgment to the vendor A. At the reception of this message, A checks if tokens are locked in the contract with the correct information. If these values are correct, and there is enough locked token, A can start selling data to B. Otherwise, it can immediately stop the subscription. For instance, if the funds F are smaller than the price P times the closing window W, B does not have enough tokens to pay for the closing window, so A has no interest in sharing data with it. At the ends of the handshake, A and B have opened a SUPRA channel and have set up the payment system.

B. Data payment

Once the funds are locked in the contract and the handshake is over, A can sell data to B. In SUPRA, the two brokers share two channels to exchange messages: an unreliable off-chain channel, and a reliable on-chain channel. Since using the on-chain channel costs fees and adds delay, the first channel used to share data is the off-chain. Then, the on-chain channel is only used when the sender does not receive an acknowledgment, to be sure that the message is delivered in time. The vendor can be paid by the smart contract either by showing an acknowledgment or by showing an on-chain message.

1) Payment using data acknowledgments: SUPRA acknowledgments contain the A's ID, and also the topic ID.

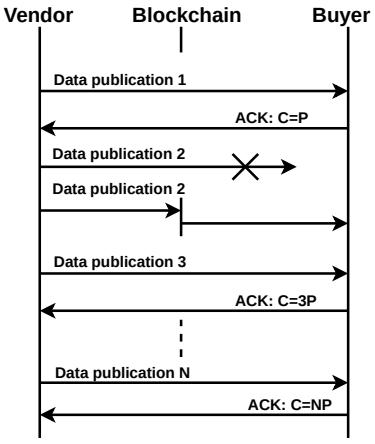

Fig. 4: Payment promises through acknowledgments

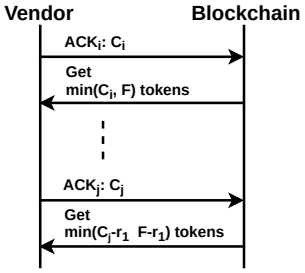

Fig. 5: The publisher claims two times tokens from the contract

For our payment system, we add a new field in the acknowledgments: the current cost C of the subscription.

As presented in Figure 4, if P is the price for each data, each acknowledgment B increases the cost C by P. If some messages go through the blockchain, because they are not acknowledged in time, the next acknowledgment includes the price of these missing data. In section IV, we explain what happens if users try to deviate from the protocol. For instance, if B purposefully does not send acknowledgments or updates the price incorrectly.

The acknowledgment has two purposes. First, like basic acknowledgments, it proves that data was received correctly. Second, since acknowledgments are signed in SUPRA, the signer cannot deny the event acknowledged and the data inside the acknowledgment. By adding the price in the acknowledgments, it proves that B is willing to give C tokens to A for the specific topic ID T. The acknowledgment contains all the information needed by the smart contract to send tokens to the vendor.

At any point in time, the vendor A can claim tokens in the contract. To do so, it has to present an acknowledgment from the user B to the smart contract, as illustrated in Figure 5. For instance, it can claim its token after i_1 messages, after i_2 messages, ..., after i_k messages, $i_1 < i_2 < \ldots < i_k$. For all j, let ACK_j be the acknowledgment for the message M_j. C_j is the cost associated with ACK_j *i.e.*, $C_j = P \times j$. For the first claim, A presents ACK_{i_1} to the smart contract

and receives $r_1 = \max(0, \min(C_{i_1}, F))$ tokens out of the F tokens locked in the contract. A cannot claim more tokens than the F locked in the contract and the claim must be non-negative. The smart contract stores in a variable R all the tokens that are already claimed by A, here $R = r_1$ so that, when the publisher later presents ACK_{i_2}, the publisher receives $r_2 = \max(0, \min(C_{i_2} - R, F - R))$ tokens. Again, the smart-contact stores $R = r_1 + r_2$, in case of future demands from the publisher. In general,

$$r_k = \max\left(0, \min\left(C_{i_k} - R, F - R\right)\right) \text{ where } R = \sum_{j=1}^{k-1} r_j$$

Hence, the total amount of tokens redeemed by the publisher, after i_k messages, is $\min(C_{i_k}, F)$, which is exactly the amount of token earned by sending i_k data (and no more than F tokens can leave the contract in total).

Claiming tokens periodically can be interesting for the vendor if it wants to get paid faster. For instance, A could claim every day its earnings for all subscriptions, but, if it wants to the minimize the transaction fees, it has to wait for an acknowledgment with a value C as close as possible to F.

It's important to notice that the public keys are registered in the distributed ledger. This means that, when someone presents an acknowledgment, the smart-contract can verify if the acknowledgment is from the correct user, B in our case, and if the entity claiming tokens is the one indicated when the tokens were locked. Hence, even if a malicious entity can get the acknowledgment, only A can claim the tokens for this specific subscription.

2) Payment using on-chain messages: Like we said previously messages can be lost when users use the off-chain channel. Which means that the acknowledgments can also be lost. In that case, we need to implement a method for the vendor to reclaim tokens without presenting an acknowledgment.

To be paid for the messages up to message M_j, we allow the vendor to present to the smart-contract a previous message M_i, its acknowledgment ACK_i, a message M_j present the ledger, and the handshake use to set up the channel. If all the messages M_x, where $i < x \leq j$, are present in the ledger, then the smart-contract uses the price indicated in the handshake and the cost in ACK_i to compute the earnings up to the message M_j.

Also the smart-contract can check the integrity of the request. Indeed, during the handshake, the brokers agreed on a value W as the closing window. Since the vendor presents the handshake, the smart-contract can learn the value W and check if the chain of messages in the ledger $M_{i+1}, ..., M_j$ is not larger than W.

Also, when the vendor uses this method, the tokens are not transferred immediately to its wallet, because malicious vendors could use this technique to claim more tokens once the subscription is closed. In Section IV, we define this delay and explain how the smart-contract can detect such behaviors.

C. Closure

There are two steps to stop the payment system: closing the SUPRA channel and claiming the remaining tokens in the contract. These steps can be done in any order. The smart contract ensures that the publisher and the subscriber leave with the right amount of tokens.

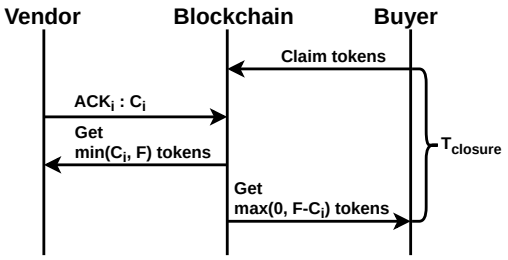

Fig. 6: The subscriber reclaims its remaining tokens

1) Closing the SUPRA channel: In SUPRA, A and B can close the channel whenever they want. For instance, B can stop the channel if its subscribers are not interested in the topic anymore, or A can stop the channel because the topic is no more shared by the publisher. To close the channel, the initiator sends a message to notify the other broker and waits for its acknowledgment. Since these messages go through the off-chain channel, they can be lost. In this case, the initiator sends the notification directly in the blockchain to ensure it arrives before a delay $T_{acknowledged}$, like other data messages.

All messages are chained together by repeating the signature of the previous message. If A stops the subscription, the notification repeats the signature of the last published data, but it is not the case if the closing originates from B, the broker on which the subscriber is connected. As said earlier, from the message chained in the stoppage notification, B will pay, at most, W data. This is to pay the potential message on the link sent by A but not yet acknowledged by B.

2) On-chain closing of the payment channel: The payment channel must be closed on-chain in order to reset the variable R and to avoid previous acknowledgments from being used again (see the proof of Lemma 4) It is closed either by the vendor or by the buyer.

The first method is to send the unsubscription message and the acknowledgment to the smart-contract. If A, the vendor, initiates the unsubscription, these two messages contain the final value for C. The smart-contract can use this value to send the final payment to A and transfer the remaining tokens to B. If B initiates the unsubscription, we need to take into consideration the window W. In this case, the smart-contract waits for a delay $T_{closure}$ before doing the final payment, to let A claim the tokens for W. To do so, $T_{closure}$ has to be superior to $\Delta_{on-chain}$, the maximum delay to add a transaction in the ledger. We will explain in Section IV, what happens if users try to cheat with W.

The second technique to close the payment channel is for B to claim the tokens in the contract, as illustrated in Figure 6. This can be used in case A leave the system, so that B can always recover its tokens. When it requests the tokens in the contract, to prevent it from recovering tokens that were intended to A, the tokens are on hold during a delay

$T_{closure}$. During this delay, A can claim the earned tokens in the contract one last time.

After a delay $T_{closure}$, B receives the remaining tokens in the contract. Once the smart contract unlock these tokens, A has no more guarantee of getting paid. If the SUPRA channel is still open, A can still share data with B, but it will not get tokens for those new messages.

If no-one closes the payment channel, then the vendor is at risk if they open a new subscription. Indeed, if the value R is not reset, then the amount of tokens the vendor can claim will be wrong.

IV. Security

In this section, we prove that our payment system provides the properties of an atomic swap [5]. This means that, if some users are malicious, correct users do not end up worse off.

In the remaining, we say that a user *can prove* some property P if it can generate a signed message, so that the judge smart contract can verify that P is true. This implies that the proof cannot use external or private information.

Before presenting the possible malicious behaviors of seller or the buyer, we will define our penalty system: if A, the seller, can prove that B, the buyer, misbehaves, it can directly claim the F tokens in the contract. Conversely, if B can prove that A misbehaves, it can directly reclaim its F tokens. When one user accuses another of misbehaving, the funds F in the contract are locked for both users, until the resolution of the conflict. Since our system uses SUPRA, the users have to respect the properties of the SUPRA channel: published data are delivered before $T_{acknowledged}$. We already explained in section II-C what happens in case of conflicts on this property. In this section, we focus on malicious behaviors in the data payment system.

Lemma 1. *Let M_i be the first message such that the value C_i of the acknowledgment ACK_i is not equal to $i \times P$. Then the seller A can prove that B made a mistake.*

Proof. Let ACK_j be the last acknowledgment received before ACK_i. If no data message where acknowledged before M_i, $ACK_j = ACK_0$ the acknowledgment for the subscription acceptation. There are two possible cases:
Case (a): $j = i - 1$. In this case, the tuple (M_j, ACK_j, M_i, ACK_i) is a proof that B made a mistake. Indeed, the judge smart contract can verify that M_i follows M_j (by checking that the previous signature of M_i is the signature of M_j) and the cost C_i is not equal to $C_j + P$. Since, $C_j = j \times P$ by assumption, we have a proof that $C_i \neq i \times P$.
Case (b): $j < i - 1$. This means that each message M_k with $j < k < i$ is either on the blockchain or stored by A (because unacknowledged). Let $Store$ be the set of stored messages by the seller A, and Txs be the set of transactions in the blockchain where messages M_k, with $j < k < i$, are published. Hence, the tuple

$$(M_j, ACK_j, Store, Txs, ACK_i)$$

is a proof that B made a mistake. Indeed, the judge smart contract can see that the cost associated with M_j is $j \times P$, by assumption, that there are $i - j$ messages correctly chained by signature by A (either in $Store$ or in Txs), and that B acknowledged the last one, which implicitly acknowledges all the previous messages. So the buyer is aware that there are $i - j$ messages so the cost associated with ACK_i should be $i \times P$. If it is not, the tuple is a proof that buyer made a mistake. □

We assumed that the data price has a constant value P, but this proof also works if the price evolves based on the message's timestamp since all the messages between M_i and M_j, if there is any, are timestamped, and the price evolution is indicated in the handshake.

Lemma 2. *An honest vendor can always claim its earnings for the sent messages.*

Proof. If the vendor has an acknowledgment, it can present the acknowledgment to receives the correct amount of tokens. Otherwise, if the vendor does not have acknowledgments from the buyer, we proved in the previous Lemma that from two acknowledgments ACK_i and ACK_j, A can present a list of on-chain message $M_{i+1}, M_{i+2}, ..., M_{j-1}$. This means that, if an honest vendor does not receive acknowledgments, claiming tokens with the method explained in Section III-B2, where a list of consecutive messages in the ledger are presented to the ledger, will work. Meaning that an honest vendor can always claims its earnings, even without an acknowledgment. □

Notice that we cannot make the difference between a dropped message and a malicious buyer who purposefully does not send acknowledgments. For this reason, if A claims tokens without an acknowledgment, the buyer B is not penalized. Still, B gains nothing by not sending acknowledgments because it will still pay for the messages sent on-chain.

Lemma 3. *If a malicious vendor tries to claim more tokens than it should, the buyer can have a proof that the vendor is malicious.*

Proof. A malicious vendor can try to claim tokens for on-chain messages sent after the end of the channel, but this behavior can be detected and penalized. To do so, when the vendor uses this method, the token transfer only takes place after a delay, to let the subscriber the time to prove the malicious behavior. This delay can be equal to $T_{closure}$.

To prove the malicious behavior, B can present several messages. If the SUPRA channel was stopped by A, B can present the unsubscription notification M_{stop} from A used to close the subscription. The difference between the timestamp inside M_{stop} and the timestamp inside the messages on the ledger will be enough to prove that the vendor is malicious.

If the SUPRA channel was stopped by B we have to take into consideration the window W. If M_{stop} was sent after the reception of M_0, B has to send acknowledgments for the messages M_i, where $1 \leq i \leq W$. If the vendor tries to claim tokens for a message where $i > W$, B can present the message

	number of on-chain messages	maximum number of unpaid data	maximum number of data paid in excess
SDPP [10]	1 every K data	$K-1$	0
Our solution	1 if no problem occurs less than any desired value $M \geq W$ otherwise	0	W

TABLE I: Comparison between our solution and SDPP [10] in terms of on-chain messages and payment guarantees

M_{stop}, the list $M_0, ..., M_W$, and the handshake to the smart-contract. With the handshake, the smart contract can learn the value of W, and with M_{stop} it can check whether the list is correct. If the list is correct, it knows that A tried to claim tokens for an invalid message. □

It is important to notice that this malicious behavior is impossible if the buyer closes on-chain the payment channel because, if the channel is closed on-chain and no new tokens where locked by the buyer since then, the smart-contract can immediately deduce that the request from the vendor is for incorrect messages.

Lemma 4. *Let M_i be a message from a previous subscription to a topic T and ACK_i be its acknowledgment. The vendor cannot use ACK_i to claim tokens in a new subscription for the same topic T with the same buyer.*

Proof. All the exchanged messages are signed and timestamped by the sender. If the vendor reuses an old acknowledgment, the timestamp will be smaller than the timestamp of the previous on-chain closing payment channel. The smart contract can compare the timestamp and prevent the vendor from irregularly claiming tokens. □

Using the blockchain to resolve conflicts also creates by design a review system on users' behaviors. In [8], authors present a smart contract to review communications and add trust in users. In our proposition, the ledger contains the history of all conflicts. Before starting a communication with a new user, a cautious user can check all the user's conflicts and accept or refuse the communication based on the conflict history.

V. Comparison with other solutions

In this section we compare our solution with SDPP [10], which the closest solution offering similar guarantees. Table I present the most important differences in terms of blockchain usage and payment guarantees. In SDPP, payment is done with an on-chain message once every K data, so the there are at most $K-1$ unpaid data. With our solution, blockchain is not used if no problems occur. Payment also uses the blockchain but can be performed at any time, so one message is enough to be paid for an entire subscription period. In the worst case, the vendor can be forced to send W messages on-chain (or more if it is willing to). Value W is decided by the vendor and can be any value greater than 0. No data can remained unpaid and at most W data is paid by the buyer after the subscription is closed.

VI. Conclusion

Compared to classical currencies, the small payment granularity of crypto-currencies allows users to sell IoT data individually without rounding the price, but, to do so, need a data payment protocol.

To the best of our knowledge, our solution is the first to have all of the following properties: message delivery is guaranteed; data payment is guaranteed for each data; data are shared in a publish/subscribe manner; blockchain (and the cost associated with it) is only used if there is a problem (abnormal delay or malicious participants); malicious behaviors can be detected and punished by a distributed smart-contract using only publicly available information.

References

[1] Jean-Philippe Abegg, Quentin Bramas, Timothée Brugière, and Thomas Noel. Distributed publish/subscribe protocol with minimum number of encryption. In *23rd International Conference on Distributed Computing and Networking*, pages 117–123, 2022.

[2] Ken Borgendale Andrew Banks, Ed Briggs and Rahul Gupta. Mqtt version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, 2019. Accessed: 2020-06-09.

[3] Vitalik Buterin. Ethereum white paper. 2013. https://ethereum.org/en/whitepaper/.

[4] David Chen, Zhiyue Zhang, Ambrish Krishnan, and Bhaskar Krishnamachari. Payflow: Micropayments for bandwidth reservations in software defined networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 26–31. IEEE, 2019.

[5] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*, pages 245–254, 2018.

[6] Ryota Nakada, Kien Nguyen, and Hiroo Sekiya. Implementation of micropayment system using iot devices. *Journal of Signal Processing*, 25(4):137–140, 2021.

[7] Satoshi Nakamoto et al. Bitcoin. *A peer-to-peer electronic cash system*, 2008.

[8] Ji-Sun Park, Taek-Young Youn, Hye-Bin Kim, Kyung-Hyune Rhee, and Sang-Uk Shin. Smart contract-based review system for an iot data marketplace. *Sensors*, 18(10):3577, 2018.

[9] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network. *Scalable o-chain instant payments*, 2015.

[10] Rahul Radhakrishnan and Bhaskar Krishnamachari. Streaming data payment protocol (sdpp) for the internet of things. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1679–1684. IEEE, 2018.

[11] Gowri Sankar Ramachandran, Sharon LG Contreras, Bhaskar Krishnamachari, Ulas C Kozat, and Yinghua Ye. Publish-pay-subscribe protocol for payment-driven edge computing. In *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.