Distributed Publish/Subscribe Protocol with Minimum Number of Encryption

Jean-Philippe Abegg, Quentin Bramas, Timothée Brugière, Thomas Noël

Distributed Publish/Subscribe Protocol with Minimum Number of Encryption

Jean-Philippe ABEGG[1,2], Quentin BRAMAS[1], Timothée BRUGIÈRE[2], and Thomas NOEL[1]

[1]University of Strasbourg, CNRS, ICUBE, France
[2]Transchain, Strasbourg, France

Abstract

Publish/subscribe is a scalable communication method for one-to-many communications. Centralized Pub/Sub protocols rely on a centralized broker that is a trusted third party managing the subscriptions. To prevent data alteration by the broker, or by an entity outside of the system, the literature proposes E2E security solutions. The issue with this solution is that they all introduce a third party who is hardly trustable in practice.

Recently, Pub/Sub protocols have been proposed using the blockchain to manage the subscriptions and provable data deliveries. In this paper, we focus our work on one publish/subscribe protocol, called SUPRA, that uses the blockchain only when necessary.

We explain why the current version of the protocol does not respect the publish/subscribe paradigm because a subscriber has to encrypt and sign each data as many times as there are subscribers. Then, present how to update the protocol in order for each data to be encrypted and signed only once, regardless of the number of subscribers, while keeping the security properties of the protocol. The result is a publish/subscribe protocol with strong delivery guarantees and traceability that can be used to share sensitive data.

Keywords: blockchain, publish/subscribe protocol, key sharing, session synchronization

1 Introduction

The publish/subscribe paradigm is a communication model used to exchange information. In this model, the broker is a central entity in a star topology. There are different kinds of publish/subscribe protocols and we will focus on *topic based* ones [4]. In this type of protocol, the publisher sends data linked to an ID, called the topic, to the broker, and the broker forwards data to all subscribers interested in the ID. The broker introduces a loose coupling between the publisher and the subscribers. This property makes the publish/subscribe model more scalable than the request/reply model, but it can also create trust issues in the system. Indeed, in a threat model where the broker can be malicious, the publisher is never sure that the data is received by the subscribers, and the subscribers are never sure that data are coming from the publisher or if there are missing messages. This issue can discourage the usage of such a paradigm for sensitive data.

The blockchain has been used to resolve this trust issue. The first propositions force the published data to go through the blockchain. Data are signed by the publisher to avoid tampering

and to prove the identity of the author. The blockchain is used as a distributed storage system for sharing data between brokers. The issue with these solutions is it cost, due to the fees associated with blockchain transactions. The higher the number of messages in the blockchain, the more expensive it becomes for the publisher.

Recently, a publish/subscribe protocol called SUPRA [1] has been proposed. It uses the blockchain only when necessary, hence making it much cheaper to deploy while having strong guarantees. The protocol uses the blockchain to set up a hybrid channel called the unidirectional channel with on-off proof of delivery. The purpose of this channel is to exchange, most of the time, messages directly between the two ends of the communication (off-chain) and only use the blockchain when we cannot guarantee the delivery of a message otherwise. Messages going through this channel are chained together with signatures, which allows the receiver to detect missing messages. Because of the message format, if N subscribers are connected to the same topic, the publisher has to encrypt N times the message and do N signatures, one for each published data. This solution does not scale well with the number of users, because with a high number of subscribers, the workload from the signature and encryption process becomes important.

Contributions

The contributions of this paper are threefold. First, we show that the current version of SUPRA does not scale well with a high number of subscribers. This problem is also present in other solutions that rely on a blockchain. Second, we present a modification of SUPRA that reduces the complexity in the number of signatures and encryptions for the publisher. With our solution, we reduce the complexity, for each data in a given topic, from N signatures and encryptions to 1, where N is the number of subscribers to this topic. Finally, we prove that, with this modification, the protocol has strong delivery guarantees. In more detail, a publisher (resp. subscriber) that follows the protocol correctly can always defend itself against any accusation and if a publisher does not follow the protocol, the subscriber can successfully accuse it with non-delivery proof.

2 Related work

End-to-end security

End-to-end (E2E) security is a method used in [2, 3, 5, 6] to resolve security issues in publish/subscribe environments. These propositions present how the publisher and the subscriber can protect the payload from unwanted accesses and alterations. To achieve this purpose, these propositions rely on encryption mechanisms. By encrypting data, security is achieved between the publisher and the subscribers because the subscribers are the only things in the system capable of decrypting data. In [6], the payload is also signed to prove message integrity.

The critical part with this security method is the key sharing process, and all these propositions introduce a new third party to handle the keys. In [2], a policy authority has to create for the broker a re-encryption key derived from the publisher's private key and the subscriber's public key. [5] also uses a trusted third party to derived keys in the system. In [3,6], the symmetric keys used for each topic are stored in a specific node in the system. All these third parties have critical information for the system. Proving the integrity of the trusted third parties is difficult as long as you do not have total read access on what it is doing. For this reason, to resolve this issue, we think that the blockchain can be used as a trustable third party to exchange public keys.

Blockchain

Blockchain is a distributed ledger technology that was first presented by Satoshi Nakamoto in 2008 [10]. 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 broadcasted 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.

Everyone connected to the blockchain has read access to the transactions inside the blocks. If the implementation allows arbitrary data inside transactions, the blockchain can be used to resolve trust issues in publish/subscribe paradigm. Indeed, publishers can share data with subscribers through the blockchain. This idea was already presented in [8,9], where a network of brokers share publish data by adding it in a block. In this solution, every piece of data is signed and added to the blockchain. It adds trust, because data are signed, and the blockchain orders messages, but it is expensive because transactions have fees. With these fees, the network pays the node and stays alive. Since all published data go through the blockchain, the publisher has to pay fees for each new data. In this proposition, there is no usage of encryption mechanisms.

In [1], a protocol named SUPRA is presented. It is a publish/subscribe protocol using the blockchain, but it reduces the number of messages. In the best-case scenario, aside from key declarations, the blockchain is never used. We will explain in further detail how the protocol works.

3 SUPRA

In this section, we will present SUPRA and the scalability issues in the protocol. To do so, we will first present the general concepts on which the protocol runs.

3.1 General concepts

Manager/worker model

The manager/worker model is an abstraction model used to represent systems using the publish/subscribe paradigm. In this model, the worker is an entity that computes or generates data. It is connected to a manager who handles a set of workers. Workers from this set may need data from workers handled by another manager, this dependency in data can be translated into topics used in publish/subscribe protocols. The manager acts as a broker for its workers, it forwards data from other managers to the interested workers in the set. Managers are publishers and are subscribers on the behalf of their workers.

With the manager/worker model, the central broker, present in the publish/subscribe model, is removed and replaced by the managers. It creates a distributed architecture, which removes the single point of failure introduced by the central broker. If a manager goes offline, only the subscriptions in which it is involved will be stopped.

The model assumes that the workers have total trust in the manager because it is own by the same entity. It means that the manager obtains no profit from dropping messages from or to its workers. It's the manager's job to make sure that the links between it and its workers are secured from unwanted alterations. On the other hand, since the managers do not trust each other, the links between managers are the only links in the system that need to be secured with a new protocol, and it is the purpose of the unidirectional channel with on-off proof of delivery.

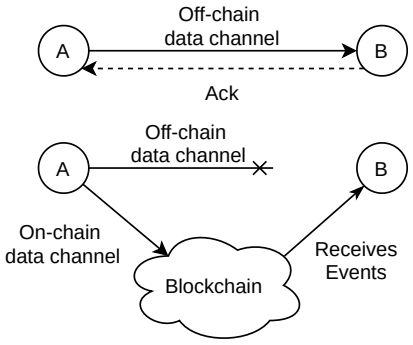

Figure 1: The two modes of communication of the unidirectional on/off chain channel protocol

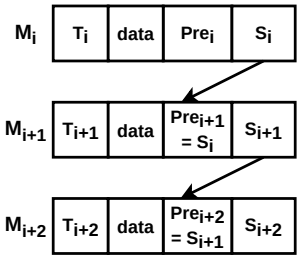

Figure 2: Three messages chained together by repeating signatures.

Unidirectional channel with on-off proof of delivery

The unidirectional channel with on-off proof of delivery is a hybrid channel used to secure the link between managers. The channel offers to the managers delivery guarantees for each message.

As presented in Figure 1, the channel is split into two sub-channels. The first sub-channel is called the off-chain channel, it is a direct link between the sender and the receiver. When messages go through this channel, the receiver has to acknowledge them. The other sub-channel is called the on-chain channel because messages go through the blockchain to arrive at the receiver. To work, the hybrid channel assumes that the sender and the receiver are reliably connected to the blockchain. This is possible by being a full node in the blockchain network, or by being connected to a trusted full node.

Before using the channel, the sender and the receiver agree on a timeout $T_{acknowledged}$. The purpose of the channel, for the sender, is to be sure that the message M_i was received before $T_{acknowledged}$. Each message M_i has a timestamp T_i. To avoid paying fees, the sender uses first the off-chain channel to send M_i directly to the receiver. This sub-channel is unreliable, so M_i or ACK_i, the acknowledgement for M_i, can be lost. Let $\Delta_{on-chain}$ be the maximum amount of time to add a transaction in a block. After a delay $T_{off-ack} = T_{acknowledged} - \Delta_{on-chain}$ from the first sending, if the sender did not receive ACK_i, the sender will use the on-chain channel to deliver M_i in time. The receiver will always have a proof of delivery for M_i, because it either has ACK_i from the receiver or the message is publicly available in the ledger. Acknowledgments are signed by the receiver but are never send through the on-chain channel.

The channel also gives guarantees for the receiver. Indeed, the message M_i is signed by the sender. The signature S_i of the message M_i proves the identity of the author but also the integrity of the message. The channel assumes that the public keys of the sender and the receiver are known. Also just like blocks in a distributed ledger, each message M_{i+1} has a reference to the previous message M_i. This is represented in Figure 2. M_{i+1} contains the signature of the previous message $Pre_{i+1} = S_i$. Chaining the messages allows the receiver to detect any missing message. At the reception of M_i, if the receiver detects a missing message, it does not send ACK_i until the missing messages are found, because each message has a reference to the previous message, so

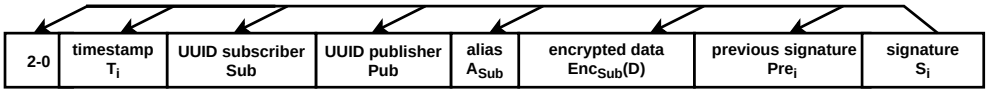

Figure 3: Data publication for the subscriber i

acknowledgments are cumulative.

Compared to counters, signatures evolve randomly. By comparing two messages using counters, a third party can guess how many messages were exchanged by comparing the counters. This action is impossible using signatures, since from one signature, it is impossible to guess the next one.

3.2 Overview of SUPRA

SUPRA is the first protocol using the manager/worker model and the unidirectional channel with on-off proof of delivery. The purpose of the protocol is to set up and use the channel in a publish/subscribe environment.

Before using the protocol, each manager has to declare a public key in the blockchain (doing so the blockchain also acts as a Public Key Infrastructure). This public key can be updated later, for instance, if the integrity of the private key is at risk. These keys are used to sign the chained messages going through the channel, but also to encrypt the data payload, and avoid unwanted accesses and alterations. The sender signs messages with its public key and encrypts data with the public key of the receiver. Each manager has a unique identifier. This identifier is in the transaction that contains the public key and is also repeated in the messages exchanged with other managers. When a message is received, the receiver uses the identifier of the source to check in the ledger the last public key declared by the sender and check the signature.

To set up a unidirectional channel, the publisher and the subscriber do a triple handshake, in which the subscriber indicated the topic name TN. After that, the publisher can publish data using the unidirectional channel with on-off proof of delivery. If the publisher does not deliver the messages in time, the subscriber can use a smart contract to prove that the publisher did something wrong.

Let M_j be the last received message, with signature S_j, from the subscriber. When it receives a new message M_i, if Pre_i is not equal to S_j, then the subscriber knows that at least one message is missing. Because of network delays, messages can be disordered but if the publisher waits for $T_{acknowledged}$ and is still unable to find the missing message, it knows that the publisher did not respect the properties of the channel. Indeed, the missing messages should at least be available in the distributed ledger. The subscriber can present M_j and M_i to SUPRA's smart contract to accuse the publisher. The publisher is unable to present proof of delivery for the missing message, since the subscriber never received them, and is penalized.

The publisher is always capable of defending itself if the subscriber does a fake accusation. If the subscriber is lying, the supposedly missing messages can not have been exchanged on-chain, otherwise, the smart contract easily found them in the ledger. This means that the publisher received an acknowledgment for the missing messages. Since each message has a dependency on the previous message, acknowledgments are cumulative. Let be ACK_k the last acknowledgment from the subscriber. If the publisher presents M_k and ACK_k to the judge it will prove that the subscriber is lying because $T_j < T_k$. This proves that the subscriber is lying because the timestamp inside the message is superior to the timestamp inside the message presented by the subscriber. This means that the subscriber was aware of a further state in the chain of signatures, and is lying. To defend themselves against fake accusations and because the protocol assumes that two different messages can not have the same timestamp, the managers have to store, for each active subscription, the last explicitly acknowledged message and the triple handshake.

Compared to other publish/subscribe protocols using the blockchain, SUPRA reduces the number of messages sent in the ledger. Indeed, in the best-case scenario, the managers just send one message on-chain, the public key declaration, then all the messages are exchanged off-chain.

In the worst-case scenario, all the messages are sent on-chain, which is what is done in the other propositions [8,9]. The protocol uses the blockchain as a trusted third party. Anybody can check if the ledger is correctly handled by the network of nodes. On top of the unidirectional channel, the blockchain is used to resolve conflicts between managers, and SUPRA uses a smart contract to detect managers who do not respect the protocol.

3.3 Scalability issue

The functionalities of SUPRA are split into 5 modules. The most important module is the publishing module. Once the subscription is set up, the publisher uses this module to send data to the subscribers. The most used message in the protocol is the one named "data publication" and, in this section, we will see why this message creates a scalability issue.

For the rest of the paper, we assume that there are N subscribers for the topic TN. Each subscriber is identified by a unique number Sub, with $0 \leq Sub < N$. In Figure 3, we can observe the message M_i that the publisher has to create to publish data D to the N subscribers in SUPRA. The message is the concatenation of:

- T_i: timestamps of M_i.
- Sub: the identifier of the subscriber.
- Pub: the identifier of the publisher. It is used by the receiver to retrieve the public key used to sign the message.
- A_{Sub}: the alias for the topic name TN. This value is used to identify multiple subscriptions between the same publisher and the same subscriber without using the topic name TN, just like port numbers.
- $Enc_{Sub}(D)$: data D encrypted with the subscriber public key.
- Pre_i: the signature of the previous message.
- S_i: the sender's signature of the message.

This message has three fields that generate a scalability issue with the number of subscribers N.

Indeed, M_i contains Sub, the identifier of the subscriber. This identifier is unique for each subscriber. With this information, the smart contract can understand which managers are involved in the communication. Since each identifier is unique, the publisher has to create N version of M_i to share the same data D.

To avoid unauthorized entity to have access to D, the publisher encrypts data. In the current version of SUPRA, the publisher uses the public key of Sub to encrypt D. Each subscriber links a public key to its identifier in the distributed ledger. The public key of each subscriber is unique. It forces the publisher to do N encryption operations for D and it creates N versions of $Enc_{Sub}(D)$, one for each subscriber.

A_{Sub} is used to identify active subscriptions between a publisher and its subscribers. This value is set during the triple handshake between the subscriber and the publisher and is chosen by the subscriber. A_{Sub} is coded on 2 bytes, which means that it is very unlikely that subscribers to a given topic have the same alias. At the same time, it can happen that two subscribers to different topics chose the same alias, which is something we want to avoid (with a numerical analysis[1], if more than 302 subscribers choose a value independently and uniformly at random, there is a probability more than $\frac{1}{2}$ that at least two will choose the same).

Because of these three fields, the value of S_i cannot be equal in each subscription to the same topic TN, and by extension the value Pre_i. Since there is N versions of the message M_i, the

[1]It is known that the probability that at least a collision occurs is $1 - \frac{2^{16}!}{(2^{16}-N)! \cdot 2^{16N}}$ (known as the Birthday problem), which is greater than $\frac{1}{2}$ when $N \geq 302$.

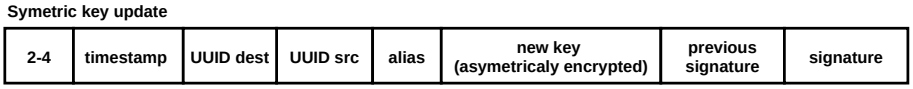

Figure 4: Symmetric key sharing message in SUPRA

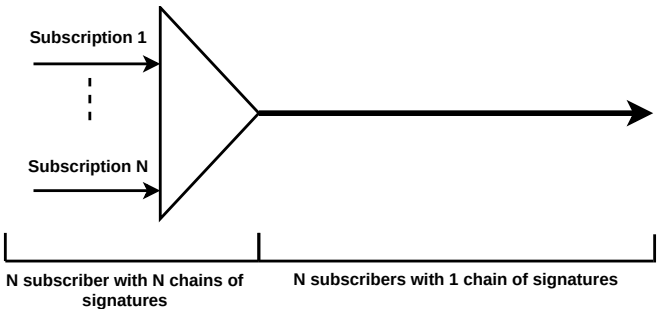

Figure 5: Schematic representation of signature synchronisation.

publisher has to do N signature operations. This idea is against the publish/subscribe model. In this model, the publisher and the subscriber are loosely coupled, they do not know each other, and the publisher should not do actions specific for each subscriber. In the current version of SUPRA, the publisher has to generate N messages for the same data, encrypted N times, and sign each version. For these reasons, we consider that the protocol does not scale well with the number of subscribers and does not respect the publish/subscribe model.

To make SUPRA scalable with the number of subscribers, we will present how to synchronize the signatures between subscriptions for the same topic.

4 Signature synchronisation

In this section, we present how SUPRA can be modified to reduce the number of signatures and encryption operations to 1 for each data, no matter the number of subscribers. We showed in the previous section that

$$M_i = T_i||Sub||Pub||A_{Sub}||Enc_{Sub}(D)||Pre_i||S_i$$

is different for each subscriber Sub (despite the fact that the data D is the same).

A schematic idea of our solution is presented in Figure 5. If we want to make SUPRA scalable, we have to find a solution to make S_i equal for all subscribers. This means that from N chains of signature, the publisher can synchronize the subscriptions to the same chain of signature. To do so, we need to prove two things:

- R1: The first message M_1 after the synchronization is the same for every subscriber.
- R2: if the message M_i is the same for every subscriber, then the message M_{i+1} will also be the same for every subscriber.

4.1 Solution Details

Removing the Sub field from the message Sub is the unique identifier of the subscriber. To resolve our scalability issue, we are forced to remove this field from the data publication. Indeed, if we let this field in the message, two signatures for two subscribers must be different. In section 4.2, we prove that removing this field from the message does not affect the traceability of the protocol and that the judge smart contract can still work without this information.

Using a symmetric key to encrypt the data As long as data are asymmetrically encrypted, it is impossible to synchronize the subscriptions, because each public key creates a different value for $Enc_{Sub}(D)$. In Figure 4, we present a new control message. This message allows the publisher to securely share a symmetric key K with the subscriber. The symmetric key is encrypted with the public key Sub. By sharing the same symmetric key K with all subscribers, the publisher only has to encrypt the data D one time by using the key K.

Let the publisher chose the topic Alias In the previous triple handshake (that occurs when opening a subscription), the subscriber chooses the value A_{Sub}. Now, to make sure that this value is equal for all subscriptions to the same topic, we have to update the handshake and let the publisher choose it. We will refer as A_{Pub} the alias chosen by the publisher for the topic TN.

The new message format for a data publication is

$$M_i = T_i||Pub||A_{Pub}||Enc_K(D)||Pre_i||S_i.$$

Since the message does not contain information about the subscriber, if the publisher can create one version M_i that is the same for all subscribers, then the message M_{i+1} will also be the same for all the subscribers (rule $R2$). Currently, if we denote M_1 the first message that the publisher uses to share data D, Pre_0 is the signature of the last message of the handshake used to set up the subscription. This signature is made by the subscriber and each subscriber has a unique public key, so there will be N version of Pre_0.

Allow Pre signature overwrite In order for all the subscribers to start with the same message M_1, we present a new control message, named Signature overwrite $SigOver$, to set the same value for Pre_0 between the N subscribers. The message has this format

$$SigOver = \\ T_{SigOver}||Sub||Pub||A_{Pub}||S_{Over}||Pre_{SigOver-1}||S_{SigOver}$$

Sub, Pub, and A_{Pub} identify the subscription. $T_{SigOver}$ is the timestamp of the message, $Pre_{SigOver-1}$ is the signature of the previous message in the chain, and $S_{SigOver}$ is the signature of the all the previous fields concatenated.

The message contains the value S_{Over} that is used in the next data publication as the value for Pre_i. This message allows the manager to set the rule $R1$ at any time, as illustrated in Figure 6. In the figure, we can observe that M_1 uses the value indicated in the signature overwrite as Pre_0 and not S_{N+1}. Before the message, the publisher has a chain of signatures with each subscriber. Then, after the publisher overwrite the Pre signature with the same value S_0 for all subscribers, the publisher ends up with one chain of signatures for all subscribers. This idea is represented in Figure 7. In this figure, we can observe that the publisher shares the same value S_3 between all subscribers. By doing so, it synchronizes the N chains of signatures into one unique chain.

The proposition can be optimized by adding the values of K and S_0, directly in the triple handshake, but it is still important to be able to do these actions outside of the triple handshake. For instance, when the publisher wants to update the symmetric key (if there is a risk of leakage), without stopping and reopening every subscription.

4.2 Security

In this section, we explain why the signature synchronization process does not change the security properties of SUPRA.

4.2.1 Reuse acknowledgements

In SUPRA, messages are acknowledged explicitly by the subscriber or implicitly when they are added in a block. If the subscriber explicitly acknowledges the message, it signs the signature

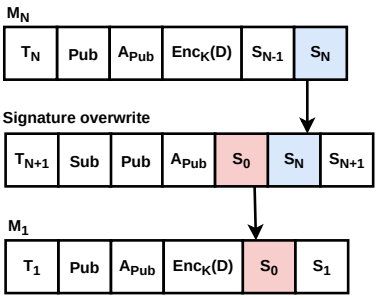

Figure 6: Example of signature overwriting

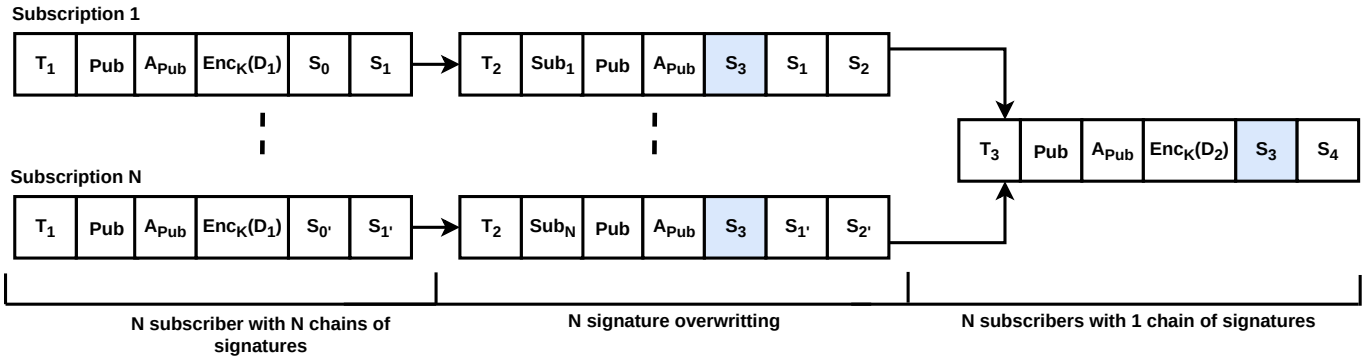

Figure 7: Synchronisation between several subscriptions.

of the message. Since we add a new message to overwrite signatures, one could ask whether the publisher can reuse an acknowledgment from the subscriber for another message on the same topic.

Recall that an acknowledgment implicitly acknowledges the previous messages on the same topic received by a subscriber. However, an acknowledgment for a message M_i should not be used directly for another message, or the same message for another subscriber.

Lemma 1. *Let M_i and $M_{i'}$ be two messages, and let $ACK_{i,k}$ be the acknowledgement of message M_i by subscriber Sub_k. Then, $ACK_{i,k}$ is not an acknowledgement of message $M_{i'}$, for any subscriber $Sub_{k'}$, with $(i,k) \neq (i',k')$.*

Proof. Assume for the sake of contradiction that $ACK_{i,k}$ is an acknowledgement for message $M_{i'}$, for a subscriber $Sub_{k'}$, with $(i,k) \neq (i',k')$.

First, if the messages have different topics, hence different topic alias, then their signatures are also different. This implies that M_i and $M_{i'}$ have the same topic alias.

Since M_i and $M_{i'}$ have the same topic alias, if $i \neq i'$, then their associated timestamp are different, say $T_i < T_{i'}$. With these different timestamps, the signatures of the two messages are different, and so are their acknowledgment.

If $i = i'$, then $k \neq k'$. In this case, by definition, $ACK_{i,k}$ is the signature by Sub_k of the signature S_i of the message M_i. Hence $ACK_{i,k}$ cannot be the signature of S_i by another subscriber $S_{k'}$ □

4.2.2 Data access on-chain

When the acknowledgment from the subscriber takes too much time, to deliver the message before $T_{acknowledged}$, the publisher sends the message on-chain. With our modifications, there is no information directly related to the subscribers inside the data publication, and the payload is symmetrically encrypted with the key K. One cannot decrypt the data without knowing this key. With our modification, we also reduce the number of messages on-chain. In the first version of the

protocol, each message M_i is different for each subscriber, so if n subscribers do not send ACK_i in time, the publisher has to send n messages on-chain. Now, since there is only one version of M_i, the publisher just has to send M_i in the distributed ledger and the n subscribers will retrieve the message.

Subscriptions can be open, but they can also be closed. A former subscriber still has the key K, which means that it can decrypt the payload of messages present in the distributed ledger, without subscribing to the topic. To avoid such a scenario, good practice from a publisher's point of view is to update the symmetrical key at each unsubscriptions. However, this can lead to a lot of updates, when subscribers come and go quickly. Another technique is to set a threshold value for the unsubscriptions, when this value is reached, the publisher updates the key. This technique introduces a trade-off between the update rate of the symmetric key and the risk for former subscribers to decrypt messages in the ledger.

4.2.3 Conflict resolution with the smart-contract

SUPRA uses a smart contract to detect publishers who do not deliver messages in time. By comparing the signatures in the last received message and a new message, the subscriber can detect if a message is missing. If the subscriber is unable to find the missing message after $T_{acknowledged}$, it knows that the publisher did not respect the protocol, because the message should at least be in the distributed ledger.

To prove that the publisher did not respect the protocol, the subscriber has to present the two messages used to detect the missing message. In our proposition, we remove the identifier of the subscriber from the message, but it is still possible for the subscriber Sub to prove that any message M_i is from an active subscription with the publisher Pub.

The publisher and the subscriber do a handshake to set up the subscription. During this phase, the publisher indicates the alias A_{Pub} used for the subscription. This value is then added to all the messages from the subscription. The handshakes of active subscriptions have to be stored by the managers. When the subscriber presents the two messages used to detect an error, it also presents the handshake to prove that there is an active subscription. The repetition of A_{Pub} and Pub in all messages proves that the messages are from the same subscription.

Lemma 2. *If a subscriber Sub has an active subscription with publisher Pub and receives two messages M_i and M_j with $i+1 < j$, and does not receive messages $M_{i'}$, $i < i' < j$ before time $T_j + T_{acknowledged}$, then Sub can accuse Pub of not respecting the protocol.*

Proof. Sub does not know how many messages are missing between M_i and M_j but by showing M_i and M_j to the judge smart-contract, everyone can see the mismatch between the previous signature Pre_j in M_j and the signature S_i of message M_i. Showing the acknowledgment of the subscription message proves that the subscription is open. If the subscription is still open, the publisher cannot show a closing subscription to defend itself. If the messages $M_{i'}$, $i < i' < j$, are not all on the distributed ledger and the publisher does not have the acknowledgments $Ack_{i',Sub}$ for those messages from Sub (which is the case since Sub did not receive those messages), then the publisher has no way to defend itself. The publisher indeed did not follow the protocol and the accusation is successful. □

4.2.4 Preventing false accusation

Lemma 3. *If a publisher Pub follows the protocol, it can always defend itself against any accusation.*

Proof. Assume that a subscriber Sub accuses Pub of not delivering a message before $T_{acknowledged}$. In the worst-case Sub has access to all the messages sent by Pub to Sub and other subscribers with other topics. To start the accusation Sub has to send a message M_j to the judge smart contract, accusing Pub of not sending the previous message M_{j-1} before $T_{j-1} + T_{acknowledged}$. Sub also sends the subscription acceptation message from Pub to show that the subscription is open. To

be valid, the Message M_j must have the same topic alias as the subscription acceptation message. Since the publisher followed the protocol correctly, this topic alias is unique and is not used for another topic so the message M_j is indeed addressed to Sub.

First, if the subscription is closed (correctly), the publisher either has an acknowledgment, from Sub, of the closing subscription message, or this message is in the distributed ledger. In this case, the publisher can defend itself.

Otherwise, if the subscription is still open, for every message M_i older than $T_{acknowledged}$, the publisher, which follows the protocol, either has received an acknowledgment or the message has been included in the distributed ledger. In both cases, it can defend itself against the accusation. □

5 Conclusion and future works

The loose coupling property of the publish/subscribe paradigm makes this communication model interesting for one to many communications. On the other hand, the broker introduced by the model creates threats that need to be handled before exchanging sensitive data through it. The propositions relying on E2E security present how data can be protected from unwanted access and alteration, even from the broker. Unfortunately, it is difficult to present trust guarantees in the third party used to add security between the publisher and the subscriber.

The blockchain, which is by design decentralized, can be used as a trusted third party in a publish/subscribe system. This is the case in SUPRA, a publish/subscribe protocol using the blockchain. However, the protocol is not scalable with the number of subscribers, based on the publish/subscribe model. In this paper, we explain how SUPRA can be updated to be scalable with the number of subscribers while keeping the security in the system.

Compared to other publish/subscribe protocols with blockchain usage, SUPRA reduces the number of fees for the users. In future work, we want to find a method to make the solution profitable for the publisher. Currently, the publisher has to pay fees for each message send on-chain. By using ideas from [7], we want to propose a method for the publisher to sell data to the subscribers, and reduce the impact of the transaction fees.

References

[1] Jean-Philippe Abegg, Quentin Bramas, Timothée Brugière, and Thomas Noël. Supra, a distributed publish/subscribe protocol with blockchain as a conflict resolver, 2021.

[2] Cristian Borcea, Yuriy Polyakov, Kurt Rohloff, Gerard Ryan, et al. Picador: End-to-end encrypted publish–subscribe information distribution with proxy re-encryption. *Future Generation Computer Systems*, 71:177–191, 2017.

[3] Markus Dahlmanns, Jan Pennekamp, Ina Berenice Fink, Bernd Schoolmann, Klaus Wehrle, and Martin Henze. *Transparent End-to-End Security for Publish/Subscribe Communication in Cyber-Physical Systems*, volume 1. Association for Computing Machinery, 2021.

[4] Patrick Th Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[5] Sam Kumar, Yuncong Hu, Michael P. Andersen, Raluca Ada Popa, and David E. Culler. Jedi: Many-to-many end-to-end encryption and key delegation for IoT. *Proceedings of the 28th USENIX Security Symposium*, pages 1519–1536, 2019.

[6] Shrideep Pallickara, Marlon Pierce, Harshawardhan Gadgil, Geoffrey Fox, Yan Yan, and Yi Huang. A framework for secure end-to-end delivery of messages in publish/subscribe systems. In *2006 7th IEEE/ACM International Conference on Grid Computing*, pages 215–222, 2006.

[7] R. Radhakrishnan and B. Krishnamachari. Streaming data payment protocol (sdpp) for the internet of things. pages 1679–1684, 2018.

[8] Gowri Sankar Ramachandran, Kwame-Lante Wright, and Bhaskar Krishnamachari. Trinity: A Distributed Publish/Subscribe Broker with Blockchain-based Immutability. pages 1–8, 2018.

[9] Gowri Sankar Ramachandran, Kwame Lante Wright, Licheng Zheng, Pavas Navaney, Muhammad Naveed, Bhaskar Krishnamachari, and Jagjit Dhaliwal. Trinity: A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence. *ICBC 2019 - IEEE International Conference on Blockchain and Cryptocurrency*, pages 227–235, 2019.

[10] Satoshi. Bitcoin: A peer-to-peer electronic cash system. pages 1–9, 2008.