UNIVERSITÉ DE STRASBOURG

ÉCOLE DOCTORALE MATHÉMATIQUES, SCIENCES DE L'INFORMATION ET DE L'INGÉNIEUR (MSII)

Laboratoire Icube - UMR7357

THÈSE présentée par:

Jean-Philippe ABEGG

soutenue le : **21 octobre 2022**

pour obtenir le grade de: **Docteur de l'université de Strasbourg**

Discipline/ Spécialité: **Informatique**

Security and efficiency of blockchain technologies applied to Internet-of-Things application

THÈSE dirigée par :	
M. NOËL Thomas	Professeur, Université de Strasbourg
THÈSE co-encadrée par :	
M. BRAMAS Quentin	Maître de conférences, Université de Strasbourg
RAPPORTEURS :	
M. LAFOURCADE Pascal	Maître de conférences, Université Clermont-Auvergne
Mme POTOP-BUTUCARU Maria	Professeure, Sorbonne Université
AUTRES MEMBRES DU JURY :	
Mme CAILLOUET Christelle	Maître de conférences, Université de Côte d'Azur
M. LUDINARD Romaric	Maître de conférences, IMT Atlantique
Mme MITTON Nathalie	Directrice de recherche, INRIA Lille
M. VALOIS Fabrice	Professeur, INSA Lyon

Abstracts

Résumé

Cette thèse traite de l'utilisation de la Blockchain dans les applications liées à l'Internet des Objets (IoT). L'IoT est un domaine présent dans notre quotidien et les applications IoT reposent sur l'utilisation d'objets contraints, des appareils informatiques beaucoup moins puissants qu'un ordinateur. L'utilisation de ces objets contraints force la définition de nouveaux protocoles et l'utilisation de nouveaux outils pour assurer les communications et la sécurité des applications. Dans cette optique, nous nous posons la question de savoir si la blockchain, une technologie de registre distribuée, peut être un outil permettant de répondre à ce problème de sécurité.

Pour répondre à cette question, nous avons réalisé deux contributions: un algorithme de consensus réduisant la consommation énergétique des nœuds blockchain, et un protocole publish/subscribe fournissant des garanties de livraison des données.

Mots clés : blockchain, internet des objets, publish/subscribe, algorithme de consensus, paiement des données

Abstract

This thesis presents how the blockchain technology is used in Internet-of-Things (IoT) applications. IoT is something present in our everyday life and the IoT applications rely on constrained devices, devices far less powerful than a computer. Using these constrained devices forces us to define new protocols and use new tools to ensure security in these applications. In this thesis, we ask ourselves if the blockchain, a distributed ledger technology, can be a tool used to increase security in IoT applications.

To answer this question, we propose two contributions : a consensus algorithm reducing the energy consumption of the blockchain nodes, and a publish/subscribe protocol presenting data delivery guarantees.

Keywords : blockchain, internet-of-things, publish/subscribe, consensus algorithm, data payment protocol

Acknowledgments

This manuscript is the result of three years of work and, over those years, I had the chance to have many encounters. In this section, I will take my time, and do my best, to thank all the people who supported me and worked with me.

First, I want to thank the jury who evaluates my work: Christelle Caillouet, Pascal Lafourcarde,Romaric Ludinard, Nathalie Mitton, Maria Potop-Butucaru, and Fabrice Valois. I thank them for accepting to judge my work.

Next, I want to thank my supervisors. Thank you to Quentin Bramas. I know that I am your first supervised PhD and, from my point of view, you did a terrific job. I will miss our discussions over blockchain problems. Also, from now on, I will try my best to write "cannot" instead of "can not". Thank you Thomas Noël. You are the perfect counterpart for Quentin, canalizing his enthusiasm and always looking at the bigger picture to orientate our efforts in the best direction. Aside from a global pandemic and a company liquidation, working with you both was pleasant. I hope that it was the same for you.

I want to thank Transchain. It seems weird to thank a closed company but I would never have started a PhD if Timothée Brugière and Pierre Banzet did not give me this opportunity. For this, I want to thank them. Also, I would like to thank all my former coworkers at Transchain. In particular, I thank Alexandre Hag, Lisa Heindrich, Jonathan Langlois, and Robin Turon-Lagot.

Thanks to all the permanent members of the Network Research team: Stéphane Cateloin, Pierre David, Anissa Lamani, Pascal Mérindol, Julien Montavont, Cristel Pelsser, Guillame Schreiner and Fabrice Theoleyre. The discussions shared with you during coffee breaks or book clubs are priceless for me because, aside from discovering topics outside the subject of my PhD, I observe different visions on how to question and present things.

Also, I thank the many PhD students on the team. Thank you Jean-Romain. What a journey since the internship in L3 ! I hope that we will make one day this publication on B^2GB. Also thanks to finding the acronym "SUPRA". Thank you Thomas A., for those breaks where we just talked about animes with you and JR. Thank you Amaury and Farzad, I wish you the best of luck for your last years of PhD. Thank you Thomas H. and keep practicing dancing.

Thanks as well to the former PhD students. Thank you Amine for bringing the chessmania to the PhD student room. By the way, if you are reading these words, Amine, your chess board is still in the PhD room. Thank you, Loic for all the discussions about football. Maybe this year Liverpool will win. Thanks as well to Renato, you are the chilliest guy I have met. Thanks as well to Sebi and Julian. Always there to explain me how to make a good Mate and correct my Spanish.

Lastly, I want to thank my sisters, Adeline and Marie-Céline, and my friends for their support during this 3 years journey.

Resumé

1 Introduction

La blockchain est une technologie de registres distribués présentés par Satoshi Nakamoto en 2008 dans le papier Bitcoin [1]. L'objectif de Nakamoto était de créer un système de paiement électronique entretenu par un réseau de nœuds sans utiliser d'autorité centrale. Les nœuds résolvent un consensus pour avoir la même copie locale du registre (les mêmes transactions, dans le même ordre). En appliquant les mêmes transactions, les noeuds possèdent les mêmes valeurs pour les comptes des utilisateurs, mais veci n'est en réalité qu'un cas d'utilisation de la blockchain.

Très vite, un cas d'utilisation plus général de la blockchain est apparu. En effet, il est possible d'utiliser une blockchain comme d'un environnement d'exécution transparent d'un programme, les smart-contracts. Les transactions du registre deviennet alors des déclarations de codes sources et des appels de fonctions. En appliquant les transactions dans le même ordre, les noeuds exécutent le code de la même manière et possède les mêmes valeurs pour les variables du programme. Ce cas d'utilisation plus général de la blockchain crée une nouvelle manière de créer de la confiance, et donc d'augmenter la sécurité, entre les utilisateurs dans des environnements distribués.

Cette confiance dans le contenu de la blockchain repose sur deux propriétés. La première, le contenu de la blockchain est public. Il est impossible pour un noeud du réseau de cacher une transaction dans un bloc aux autres noeuds du réseau. Ils ont tous la même copie du registre. La deuxième, la donnée ne peut pas être modifiées. Tant que certaines conditions sont respectées sur la quantité de noeuds honnêtes dans le réseau, il est impossible pour un noeud malveillant de modifier une transaction dans un bloc, une fois que ce dernier est ajouté dans le registre. On dit que la donnée est immuable. Grâce à ces propriétés, les utilisateurs de la blockchain peuvent avoir confiance dans le contenu du registre.

Dans cette thèse, nous nous posons la question de savoir si la blockchain peut être un outil intéressant pour augmenter la sécurité dans un environment distribué en particulié, l'Internet des Objets (IoT). L'IoT est un domaine présent dans notre quotidien. Il peut être utilisé dans un cadre personnel avec par exemple l'utilisation de montres connectées ou de maisons connectées, mais aussi dans un cadre industriel avec par exemple les usines connectées. Les applications IoT reposent sur l'utilisation d'objets contraints, des appareils informatiques beaucoup moins puissants qu'un ordinateur, pour mesurer un environnement réel.

La difficulté de faire cohabiter l'IoT et la blockchain vient de la différence de puissance entre les objets IoT et la puissance nécessaire pour être un membre du réseau blockchain. Les objets contraints ne sont pas assez puissants pour être directement intégrés dans le réseau blockchain. Il est donc impossible de simplement ajouter tous les objets connectés dans le réseau blockchain pour bénéficier des avantages du registre distribué. Ce problème peut être résolu de deux manières. La première est de réduire les ressources nécessaires pour être un nœud blockchain pour sur le long terme envisager d'ajouter des objets contraints en nœuds blockchain. La deuxième solution est d'utiliser plus intelligemment la blockchain, en se basant sur des architectures d'applications IoT existantes et en observant où peut être utilisé la blockchain et comment elle peut améliorer la sécurité des systèmes.

Pour répondre à notre question sur l'utilité de la blockchain dans le domaine de l'IoT, nous avons réalisé deux propositions. La première est un protocole publish/subscribe utilisant la blockchain, SUPRA. Le paradigme publish/subscribe est un modèle de communication utilisé en IoT. Notre deuxième proposition est un algorithme de consensus demandant moins de puissance de calcul pour les participants, la Preuve d'Interaction.

2 SUPRA

Le modèle *publish-subscribe* est un paradigme de communication. Dans ce modèle, il existe trois entités:

- le *publisher*, qui génère des événements, dans notre cas, des données.
- le *subscriber*, qui souhaite recevoir des événements.
- le *broker*, qui est un intermédiaire entre les *publishers* et les *subscribers*.

Les *pubishers* et les *subscribers* ne se connaissant pas directement. Ils n'ont pas besoin d'être connectés en même temps pour partager de l'information. Ce modèle est alors plus économe en ressources et s'adapte mieux à des communications entre de nombreux noeuds que le modèle *request-reply* [2]. Pour ces raisons, ce modèle est utilisé dans des applications IoT.

Il y a différents types de protocoles *publish-subscribe* et nous allons nous concentrer sur les protocoles *publish-subscribe* utilisant les *topics*. Un *topic* est un identifiant, representé par une chaîne de caractères, associé à la donnée crée par le *publisher*. Dans ces protocoles, le *subscriber* annonce son intérêt pour les données associées à un *topic*. Lorsqu'une nouvelle donnée de ce *topic* est générée par le *publisher*, alors la donnée est envoyée au *subscriber*. Les communications de ce modèle sont unidirectionnelles, du *publisher* vers les *subscribers*.

Le *broker* est l'entité centrale de ce modèle, car elle est chargée de transférer les messages du *publisher* vers les *subscribers*. Ce tier de confiance crée des problèmes de sécurité dans le modèle et il existe plusieurs protocoles remplaçant ce broker par une blockchain [3, 4].

Travaux connexes. Trinity [3] est, à notre connaissance, le premier protocole distribué *publish-subscribe* utilisant la blockchain. Dans cette proposition, il y a plusieurs *brokers* chacun étant un nœud d'un réseau blockchain. Le *publisher* signe les données qu'il crée, et il les envoie à son *broker* (en lequel il a toute confiance). Les données seront par la suite ajoutées dans un bloc de la chaîne. Une fois le bloc crée, les autres *brokers* envoient les données à leurs *subscribers* locaux abonnés à ces données. La blockchain permet ici d'avoir une confiance complète dans la donnée et la communication. Les données sont signées par le *publisher* pour confirmer leur origine, le *broker* ne possède pas la clé privée du *publisher*. La blockchain fournie aussi un ordonnancement total et immuable pour les données.

Le problème de cette solution est l'usage intensif de la blockchain. Chaque donnée est envoyée sur la blockchain par le biais d'une transaction. Cette dernière possède un coût en jetons spécifique à la blockchain utilisée. Envoyer un grand nombre de données augmente ce coût en jetons qui doit être payé par le *publisher* ou le *broker* associé. De plus, les données sur la blockchain sont conservées durant toute la durée de vie de la blockchain. Au fur et à mesure des publications, la taille de la blockchain va croître, ce qui va augmenter les ressources de stockage nécessaires pour les noeuds.

Première version de SUPRA

SUPRA est un protocole distribué publish/subscribe utilisant la blockchain. À l'inverse des autres propositions, ce protocole essaie de réduire au maximum le nombre de messages envoyés sur la blockchain, qu'on appelle *on-chain*. Ceci afin de réduire le coût des frais de transaction et l'impact du temps de validation des transactions sur les performances du protocole. De plus, le protocole souhaite garantir, pour le publisher, la livraison des messages avant un délai T depuis le premier envoi et, pour le subscriber, l'ordre et l'origine des messages. L'intégralité des messages est horodatée et signée par la source. De plus, les clés publiques pour vérifier les signatures sont enregistrées dans la blockchain.

Pour les communications unidirectionnelles entre le publisher et le subscriber, le protocole utilise deux canaux de communication: un lien *off-chain*, qui est une connexion non-fiable entre le publisher et le subscriber (ne passant pas par la blockchain), et un lien *on-chain* qui est une connexion fiable à travers une blockchain (c'est-à-dire que le publisher, resp. le subscriber, est soit connecté de manière fiable à un nœud blockchain, soit est lui-même un nœud blockchain). Pour envoyer un message, le publisher utilise en premier lieu le lien off-chain et attend de recevoir un acquittement de la part du subscriber. L'acquittement du message est une preuve pour le publisher que le message a été reçu dans les temps. En cas de conflit, cette preuve peut être

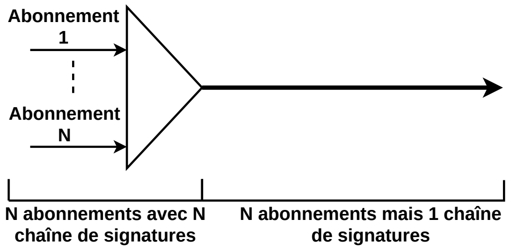

Abonnement 1
Abonnement N
N abonnements avec N chaîne de signatures
N abonnements mais 1 chaîne de signatures

Figure 1: Le *publisher* possède plusieurs abonnement actifs, possédant chacun une chaîne de signature, et il fusionne ces chaînes en une seule.

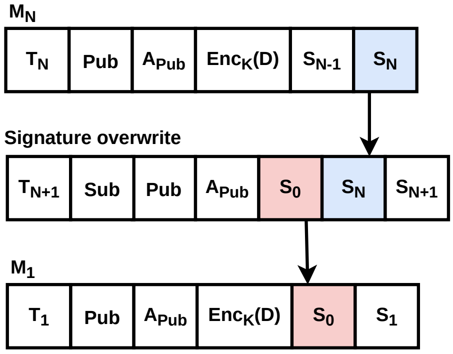


Figure 2: Le *publisher* utilise le message $SigOver$ pour choisir la valeur de Pre_i

envoyée à un tiers qui sera alors capable de vérifier la bonne réception du message en utilisant uniquement des données publiques. En l'occurrence, les clés publiques utilisées pour signer les messages sont dans la blockchain, et l'acquittement contient la signature du message acquitté.

Si, après l'envoie d'une donnée au subscriber, l'acquittement ou le message d'origine sont perdus, le publisher utilise alors le canal on-chain, c'est-à-dire, envoie le message dans la blockchain pour qu'il soit inclu dans un bloc avant l'écoulement du délai T. La présence du message dans un bloc est une preuve de sa réception par le subscriber, car le protocole suppose que les utilisateurs sont connectés au réseau blockchain. Avec ce système, on réduit le nombre de messages envoyé dans la blockchain par rapport aux autres propositions de protocole.

Pour détecter les messages perdus, ou désordonnés, chaque message envoyé par le publisher aux subscribers est chaîné au précédent en répétant sa signature. Si le message répète une signature inconnue, le subscriber détecte alors une perte. À cause du réseau, le message manquant peut être retardé, mais si le subscriber attend un délai T et n'est toujours pas capable de trouver le message manquant, alors il sait que le publisher n'a pas respecté le protocole. En effet, le message manquant devrait au minimum être présent dans la blockchain.

En montrant à un tiers (ou même un smart-contract) le message utilisé pour détecter l'erreur et le dernier message correct envoyé par le publisher, le subscriber peut prouver que le publisher n'a pas respecté le protocole. Le tiers peut vérifier que les signatures ne correspondent pas, et que les messages manquants ne sont pas dans la blockchain. Pour éviter les fausses accusations, le tiers doit accorder un délai au publisher pour présenter une preuve de réception des messages manquants : un acquittement venant du subscriber. Pour que ce processus de détection fonctionne, il est important pour le subscriber de ne pas envoyer d'acquittement avant la réception de l'intégralité des messages précédents.

Cette version du protocole possède un défaut pour partager les données à un grand nombre de subscribers. Le chaînage des messages force le publisher à réaliser N signatures si il souhaite partager la même donnée à N *subscribers*. Ceci va à l'encontre du modèle publish/subscribe où l'on souhaite avoir une faible quantité de travail pour partager l'information. Nous allons présenter une méthode pour réduire ce nombre d'opérations à 1 pour N *subscriber*.

Réduction du nombre de signatures

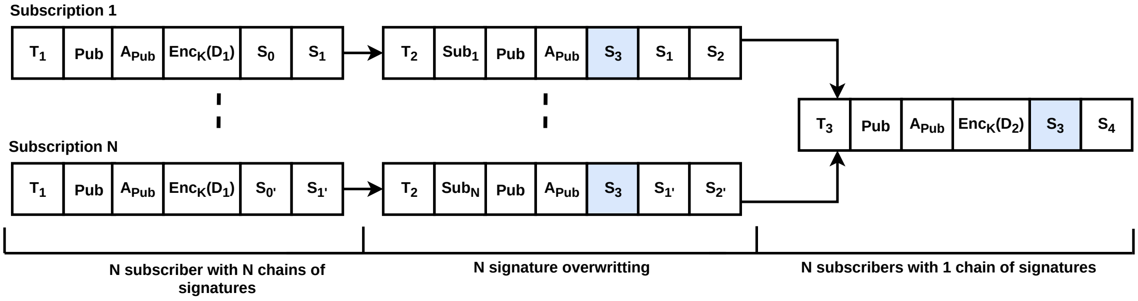


Figure 3: Le publisher unifie les signatures de publishers abonnements.

Le problème du nombre de signatures réalisées par le *publisher* vient du format de messages et des fonctionnalités du protocole. Sur la Figure 1 est représentée une version graphique de notre solution à ce problème. Nous voulons que le *publisher* soit capable à partir de plusieurs connexions avec plusieurs subscribers pour un même topic TN de créer une seule chaîne de signatures commune entre toutes les connexions. Pour ce faire, nous devons vérifier les deux règles suivantes:

- R1: après avoir unifié les chaînes de signatures, le message M_i est identique pour tous les *subscribers*.
- R2: Si le message M_i est identique pour tous les *subscribers*, alors le message M_{i+1} l'est aussi.

De ce double objectif, découle un nouveau format de message qui ne contient plus d'informations désignant un subscriber en particulier. Dans la version précédente du protocole, le message contenait par exemple l'identifiant du subscriber. Cette information est unique par subscriber et force le publisher à réaliser une signature par subscriber. Nous proposons ce nouveau format:

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

T_i est l'horodatage du message M_i. Pub est l'identifiant du publisher. A_{Pub} est l'alias choisi par le publisher (l'équivalent d'un numéro de port). $Enc_K(D)$, la donnée chiffrée par une clé symetrique. S_i est la signature du message et Pre_i est la signature du message précédent, $Pre_i = S_{i-1}$.

Ce nouveau format ne contient pas d'information identifiant un *subscriber* en particulier, ce qui signifie qu'en l'utilisant, si nous sommes capables de faire un message M_i identique pour tous les *subscribers*, alors le message M_{i+1} sera lui aussi identique (R2).

Pour résoudre le problème de signatures, nous devons maintenant faire en sorte de créer une version unique du message M_i pour tous les *subscribers* du même topic. Dans SUPRA, chaque message M_i répète la signature du message précédent dans le champ $Pre_i = S_{i-1}$. La première valeur de Pre_i est la signature du dernier message de la poignée de mains utilisée pour mettre en place l'abonnement. Cette signature est réalisée par le *subscriber*, il y a donc N versions de Pre_i quand les connexions entre le *publisher* et les *subscribers* sont mises en place.

SUPRA ne fait aucune supposition sur l'algorithme de signature. Il peut engendrer ou non des collisions. Si les collisions sont impossibles, alors il devient impossible d'unifier la valeur de Pre_i, mais, même si l'algorithme réalise des collisions, unifier la valeur de Pre_i reste une opération aléatoire et difficile.

Pour parvenir à unifier la valeur de Pre_i, quelle que soit l'hypothèse sur l'algorithme de signature utilisé, nous ajoutons un nouveau message de contrôle, et nous avons représenté son fonctionnement sur la Figure 2. Ce message se nomme *Signature Overwrite* et il possède le format suivant:

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

En utilisant ce message, le *publisher* peut partager une valeur S_{Over} à tous les subscribers. Comme représentée sur la Figure 3, cette valeur va être utilisée dans le message suivant M_i où $Pre_i = S_{Over}$. Si le publisher partage la même valeur $SigOver$ à tous les *subscribers* d'un même topic, il est alors certains d'avoir transformé les chaînes de signatures de chaque subscriber en une seule chaîne unique pour tous les subscribers.

Grâce à nos modifications dans le format de message et à ce nouveau message de contrôle, le *publisher* est capable de réduire le nombre d'opérations de signature et de chiffrement de N à 1 pour partager la même donnée à N *subscribers*.

Securité

SUPRA est un protocole qui assure une traçabilité des données. Il est toujours possible pour un utilisateur de prouver la provenance d'un message et sa destination. Cette traçabilité permet d'utiliser un smart-contract pour résoudre les possibles conflits entre les utilisateurs. C'est-à-dire, si un message du *publisher* n'est pas délivré après un délai T depuis son premier envoi. Avec notre nouveau format de message, notamment avec l'absence de l'identifiant du *subscriber*, on peut se demander si cette traçabilité est toujours garantie.

Pour assurer la résolution des conflits, SUPRA demande à ses utilisateurs de conserver la poignée de main en trois temps utilisée pour mettre en place la connexion entre le *subscriber* et le *publisher*. Dans cette poignée de main, sont échangés le nom du topic et l'alias. Dans ces messages, les identifiants du subscriber et du publisher apparaissent. En présentant cette poignée de main, il est toujours possible de prouver l'existence d'une connexion entre le *subscriber* et le *publisher*.

Pour prouver la réception d'un message, le *publisher* doit posséder un acquittement pour ce message ou le message doit être présent dans la blockchain. L'acquittement du message M_i par le *subscriber* n'est que la signature, par le *subscriber*, de S_i. Si on considère S_i' la signature acquittée par un ancien acquittement, un *publisher* malveillant peut essayer de modifier Pre_i pour que $S_i = S_i'$. En réalité, on peut montrer que cette opération est impossible, grâce à l'horodatage des messages. Le protocole présume que si M_i et M_j sont deux messages, avec $i \neq j$, alors $T_i \neq T_j$. Ce qui signifie si $S_i = S_i'$ alors $T_i = T_i'$ et donc $i = i'$. Si ce n'est pas le cas, les horodatages de l'acquittement et du faux message ne correspondent pas, et un tiers (ou un smart-contrat) peut facilement détecter cette tentative de fraude.

Vente des données

Pour l'IoT, nous avons étendu SUPRA pour permettre la vente de données en les partageant de manière *publish/subscribe*. Cette opération est possible en ajoutant le prix des communications dans les acquittements et en modifiant le handshake pour mettre en place la connexion et le smart-contract.

Avant de débuter le partage des données, le *subscriber* verrouille une quantité de tokens dans le smart-contract. Grâce aux acquittements, le *publisher* est capable de récupérer une partie de ces tokens et d'être rémunéré pour le service rendu au subscriber. Le smart-contract garanti que le *publisher* est l'unique entité capable de récupérer une partie des tokens.

Conclusion

SUPRA est un protocole *publish/subscribe* utilisant la blockchain pour ajouter des garanties sur l'origine et la livraison des données. La blockchain est ici un médium neutre permettant le partage public d'information non-critique pour mettre en place des garanties de sécurité aux utilisateurs.

La première version du protocole souffre d'un problème de passage à l'échelle, et force le *publisher* à réaliser un nombre d'opérations ne respectant pas le modèle *publish/subscribe*. Nous avons présenté le protocole SUPRA ainsi qu'une amélioration permettant de corriger ce problème, tout en gardant la sécurité de la version d'origine. Grâce à cette extension, le nombre de messages envoyés dans la blockchain est réduit au minimum. Peut importe le nombre de *subscriber* pour un même topic, le *publisher* devra envoyer uniquement un unique message dans la blockchain, si il doit envoyer un message dans le registre pour le livrer dans les temps.

3 Preuve d'interaction

Introduction La Blockchain est un protocole permettant de maintenir un livre de comptes de manière distribuée entre plusieurs participants. Les transactions sont ajoutées sous la forme de *blocs* liés entre eux pour former une chaîne. Comme tous les nœuds du réseau ont le même rôle, un système d'élection de leader est utilisé pour élire le nœud responsable de l'ajout du prochain bloc. Le protocole Blockchain le plus connu, Bitcoin, utilise la *preuve de travail* (PoW) comme système d'élection [**bitcoin**]. Avec la PoW, la probabilité qu'un nœud soit élu est proportionnelle à sa puissance de calcul. Cela incite donc les participants à dépenser beaucoup d'énergie afin d'augmenter leur chance d'être élu.

Nous proposons une alternative à la preuve de travail appelée preuve d'interaction. Puis nous montrons comment utiliser ce système de preuve pour construire un protocole Blockchain, similaire à Bitcoin, mais qui ne nécessite presque aucun travail. De plus, notre Blockchain offre une meilleure résistance aux attaques par minage égoïste.

La *preuve de travail* (PoW) est à l'origine une technique anti-spam proposée en 1999 et adaptée en 2008 par le protocole Bitcoin [1]. La PoW peut être vue comme un système d'élection de leader, pour sélectionner le nœud responsable de l'ajout du prochain bloc dans la chaine de blocs. La PoW permet de se protéger contre les attaques Sybil, les dénis de service et a un faible coût en communication, mais elle est énergivore.

Plusieurs alternatives à la PoW existent, par exemple la preuve d'enjeu (PoS) et la preuve de temps écoulé (PoET). Ces alternatives posent cependant des contraintes sur le réseau sous-jacent, ou introduisent de nouvelles failles de sécurité.

M. Abliz et T. Znati [5] ont présenté un système de protection anti-spam qui n'a pas, à notre connaissance, été utilisé dans le contexte des Blockchains. L'idée de ce système est la suivante: quand un serveur de ressources reçoit un grand nombre de requêtes, il demande au client de faire un tour dans le réseau *i.e.*, visiter un ensemble de serveurs ayant le même propriétaire que le serveur de ressources. Une fois ce tour réalisé, le client peut prouver au serveur de ressources la réalisation de ce tour pour obtenir l'accès à la ressource. La preuve d'interaction étend ce schéma pour être utilisé au sein d'une Blockchain.

Modèle

On considère un réseau N constitué de n nœuds, dont le graphe de communication est complet. Les nœuds sont identifiés de manière unique par leur clé publique (l'association clé publique – nœud est connue pour l'ensemble du réseau). Chaque message envoyé est signé par l'émetteur et ne peut pas être modifié ou falsifié par un autre nœud. Les communications sont supposées partiellement synchrones, comme dans le protocole Bitcoin. Le temps maximum d'attente avant la réception d'un message est borné par une valeur inconnue.

L'opérateur $\cdot$ désigne la concaténation. La fonction H est une fonction cryptographique de hachage. Nous écrirons $\texttt{sign}_u(m)$ la signature par le nœud u du message m. Nous supposons que la fonction de signature est déterministe et ne dépend que du message et de la clé privée de u. Cette hypothèse peut sembler forte car avec les schémas de signatures habituels, un message peut avoir plusieurs signatures différentes valides. Cependant, nous pensons qu'elle peut être levée en pratique, et nous traiterons ce problème dans des travaux futurs.

Nous allons par la suite présenter les algorithmes permettant la génération et la vérification d'une preuve d'interaction et montrer comment ces algorithmes sont utilisés pour construire un protocole Blockchain.

La Preuve d'Interaction (PoI)

On note d la *dépendance* de la preuve (dans notre cas, elle correspond au hash du bloc précédent) et m le *message* prouvé (dans notre cas, il correspond à la racine de l'arbre de Merkle qui stocke les transactions du bloc en cours). On note D la distribution de probabilité utilisée pour tirer au hasard la longueur des tours dans le réseau de chacun des nœuds. D représente la difficulté de la preuve.

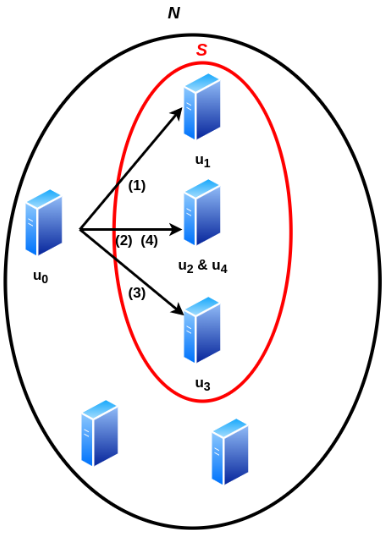


Figure 4: u_0 interagit avec un sous ensemble S de N

Calculs Préliminaires Un nœud u_0 qui veut générer une PoI doit connaître *(i)* le sous-ensemble S de nœuds du réseau avec qui il va interagir et *(ii)* la taille du tour L qu'il doit réaliser. Ces deux informations sont calculées de manière pseudo-aléatoire avec comme graine s_0, la signature par u_0 de la dépendance d ($s_0 = \texttt{sign}_{u_0}(d)$).

On a $S = \{S_0, S_2, \ldots, S_{n_S-1}\}$, un sous-ensemble de N pseudo-aléatoire de taille $n_S = \min(20, n/2)$, généré de manière déterministe à partir de s_0. De même, L est un nombre pseudo-aléatoire qui suit la distribution D, générée de manière déterministe à partir de s_0. Le nœud u_0 n'a donc aucun moyen de modifier S et L, et chaque nœud du réseau qui génère une PoI a son propre ensemble S et sa propre longueur L, supposés aléatoires. On voit sur la Figure Figure 4 que le nœud u_0 interagit avec une suite de nœuds $u_1, u_2, \ldots$ dans le sous-ensemble S.

Génération de la Preuve Pour un nœud u_0, le tour dans le réseau se réalise en L phases séquentielles. Chaque phase $j \in [1, L]$ consiste en la création d'un hash h_j, à partir du hash h_{j-1} de la phase précédente, avec $h_0 = H(s_0 \cdot m)$. A chaque phase $j \in [1, L]$, u_0 réalise les étapes suivantes: (i) calculer le nœud u_j de S avec qui il va interagir. On a $u_j = S_i$ où l'indice i est le hash précédent modulo n_S, $i \equiv h_{j-1} \mod n_S$; (ii) envoyer à u_j le tuple (d, m, h_{j-1}). Le nœud u_j répond en envoyant $s_j = \texttt{sign}_{u_j}(d \cdot m \cdot h_{j-1})$; (iii) signer s_j pour obtenir $s'_j = \texttt{sign}_{u_0}(s_j)$; (iv) calculer $h_j = H(s'_j)$.

Les phases sont exécutées de manière séquentielle jusqu'à trouver s_L et $s'_L = \texttt{sign}_{u_0}(s_L)$. La Figure Figure 5 montre un exemple de tour complet pour $L = 3$, avec pour d le hash du bloc précédent.

Definition 3.1 *En utilisant les notations précédentes, la preuve d'interaction de dépendance d, du message m par le nœud u_0, avec la difficulté D est la suite* $(s_0, s_1, s'_1, s_2, s'_2, \ldots, s_L, s'_L)$.

Vérification de la Preuve Pour qu'un nœud u vérifie une preuve $(s_0, s_1, s'_1, s_2, s'_2, \ldots, s_L, s'_L)$ venant de u_0 de dépendance d, message m et difficulté D, il doit d'abord vérifier que s_0 est bien la signature par u_0 de d, puis recalculer S, L et h_0 à partir de s_0, D et m. Par ailleurs, u peut calculer la suite des hashs $h_j = H(s'_j)$, $1 \leq j \leq L$.

Puis u doit vérifier que chaque s'_j $(1 \leq j \leq L)$ est bien une signature valide par u_0 de s_j. Pour finir, u doit vérifier que chaque s_j $(1 \leq j \leq L)$ est bien une signature valide de $d \cdot m \cdot h_{j-1}$ par S_i, où $i \equiv h_{j-1} \mod n_S$.

Protocole Blockchain Utilisant la Preuve d'Interaction

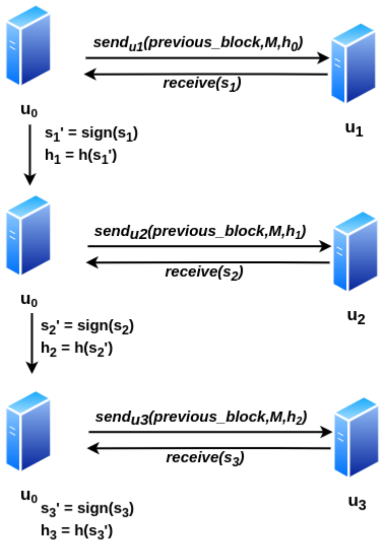


Figure 5: u_0 interagit de manière séquentielle avec les différents nœuds.

Dans cette partie, nous présentons le protocole Blockchain qui utilise notre Preuve d'interaction et en listons plusieurs propriétés intéressantes.

Génération de blocs On construit une Blockchain similaire à la Blockchain Bitcoin. Un bloc contient la liste des transactions (stockées dans un arbre de Merkle), le hash du bloc précédent, la difficulté (on suppose que la distribution D est caractérisée par un nombre, qui peut être la moyenne par exemple) et d'autres méta-données ($e.g.$, version, horodatage). Dans Bitcoin, la preuve de travail est stockée sous forme de nonce. Dans notre cas, chaque bloc contient la preuve d'interaction dont la dépendance est le hash du bloc précédent et le message est la racine de l'arbre de Merkle contenant les transactions du bloc.

Quand un bloc, de hash d, vient juste d'être ajouté à la Blockchain, chaque nœud du réseau regroupe les transactions valides non encore présentes dans un bloc, les stocke dans un arbre de Merkle de racine m, et commence à générer une PoI de dépendance d et message m. Le premier nœud qui termine sa PoI ajoute son bloc à sa chaîne de blocs et diffuse son bloc dans tout le réseau. Quand un nœud reçoit un bloc contenant une preuve valide, il ajoute ce bloc à sa chaîne de blocs. Si ce bloc augmente une branche qui devient la branche la plus longue, alors ce nœud arrête sa PoI en cours et en commence une nouvelle en utilisant ce nouveau bloc comme dépendance.

Ajustement de la difficulté La difficulté peut être ajustée comme dans le protocole Bitcoin, c'est-à-dire en utilisant les horodatages inscrits dans les blocs, de manière à ce que la durée entre deux blocs soit en moyenne une constante B. Ici, cette difficulté peut être paramétrée précisément en utilisant la distribution D. Par exemple, supposons que la durée moyenne d'une communication dans le réseau soit notée Com. Si chaque tour a une longueur choisie aléatoirement de manière uniforme entre 1 et $\lceil B/Com \rceil (n+1) - 1$, alors on peut montrer que la longueur du plus petit tour, parmi les PoI de tous les nœuds, sera en moyenne $\lceil B/Com \rceil$ (soit une durée $\lceil B/Com \rceil \times Com$ entre deux blocs).

Récompenses et Pénalités Chaque bloc rapporte une récompense à tous les nœuds ayant participé à la PoI. Cette récompense peut être définie comme dans le protocole Bitcoin $i.e.$, le nombre de *tokens* offerts aux nœuds récompensés contient les frais de transaction plus une certaine quantité, qui décroît après un certain nombre de blocs.

On suppose aussi que tous les noeuds du réseau ont verrouillé une certaine quantité de *tokens* dans la Blockchain. Ils peuvent ainsi être pénalisés s'ils ont certains comportements. Par exemple, si un nœud crée deux blocs différents, ayant la même dépendance d, alors il peut être pénalisé, comme le détaille le paragraphe suivant.

Non parallélisable Un nœud u_0 ne peut pas calculer en parallèle une PoI pour un message m et une dépendance d donnés. En effet, pour contacter le i-ème nœud u_i, u_0 doit avoir demandé une signature, qu'il ne peut pas prévoir, au nœud u_{i-1}. Cela implique que le calcul de la preuve ne peut être réalisé que de manière séquentielle en commençant par le calcul de s_0.

Pour effectuer plusieurs preuves en parallèle, un nœud u_0 doit créer plusieurs blocs (avec des valeurs m différentes). Cela permet d'augmenter ses chances d'obtenir la récompense, mais il risque alors d'être fortement pénalisé si un autre nœud s'en rend compte. Or, l'ensemble S des participants et la longueur L ne dépendent pas de m. Donc, si le nœud u_0 crée deux blocs, avec deux valeurs m_1 et m_2, il y a une forte probabilité qu'un même nœud v participe aux deux PoI du nœud u_0. Ce nœud v va recevoir deux demandes de signature, une avec la valeur m_1 et l'autre avec la valeur m_2, émanant du même nœud u_0, pour la même dépendance d. Dans ce cas, v pourra donc lancer l'alerte et à son tour gagner une récompense.

Attaque par recherche d'un tour avantageux On suppose pour cette attaque qu'un sous-ensemble du réseau est malicieux, dont u_0. Ils partagent leur clé privée dans le but d'accélérer la génération de la PoI.

Comme la suite des nœuds de S avec qui u_0 doit interagir dépend du bloc en cours, u_0 pourrait, en théorie, créer plusieurs blocs afin de choisir celui dont tous des noeuds à contacter sont malicieux. S'il trouve un tel bloc, u_0 pourrait générer la PoI sans envoyer un seul message (il connaît les clés privées des autres noeuds malicieux). Cependant, comme l'ensemble S ne dépend pas du bloc en cours, u_0 ne peut pas le modifier. Comme S est supposé être un sous-ensemble aléatoire de N, il contient la même proportion de nœuds malicieux que dans le réseau tout entier. Supposons que cette fraction soit de 10% et que u_0 doive effectuer un tour de longueur 100, alors il lui faudra en moyenne générer 10^{100} blocs avant de trouver un tour qui ne contienne que des nœuds malicieux.

Minage égoïste Le minage égoïste est un comportement bien connu dans les Blockchains utilisant la preuve de travail. Elle consiste en un groupe de mineurs malicieux qui collaborent pour maximiser leurs gains. Quand un nœud du groupe trouve un bloc il le partage uniquement avec ses collaborateurs. Le groupe de mineurs commence alors à travailler secrètement sur le prochain bloc pendant que les nœuds honnêtes gaspillent leur énergie dans la recherche d'un bloc sur une branche obsolète.

Notre protocole permet de limiter ce genre de comportement. En effet, lorsqu'un nœud u_0 effectue sa PoI, chaque demande de signature envoyée à u_i contient la dépendance d utilisée par u_0. Si u_0 trouve un bloc, il est obligé de le révéler au moins au membre de S avec qui il interagit pour le bloc suivant (on a vu au paragraphe précédent, que la recherche d'un tour ne contenant que des noeuds malicieux n'était pas raisonnable en pratique). Un nœud honnête u_i refusera de signer un bloc qu'il ne connaît pas. Cette collaboration entre les différents nœuds pour générer les preuves impose une bonne propagation des blocs dans le réseau.

Conclusion

Nous avons présenté une alternative à la PoW demandant une faible puissance de calcul pour les participants. De plus, on peut montrer que le protocole Blockchain basé sur la PoI a une complexité en nombre de messages par unité de temps linéaire en fonction de la taille du réseau. Notre protocole fait l'hypothèse que les nœuds du réseau sont connus, mais nous pensons que les techniques existantes qui permettent de lever cette hypothèse sont particulièrement adaptées à la PoI.

4 Conclusion

En conclusion, la blockchain est un outil intéressant pour améliorer la sécurité dans les applications liées à l'Internet des Objets. La transparence dans le contenu du registre et la non modification de son contenu par un utilisateur malveillant permet de partager des informations entre les utilisateurs de l'application IoT pour améliorer la sécurité. Cepedant, pour utiliser de manière réaliste la blockchain, il faut prendre en compte des propriétés importantes de la blockchain. En l'occurence, les ressources nécessaire pour être un noeud blockchain et le coût d'utilisation de la blockchain pour ajouter une transaction. C'est ce que nous avons réalisé avec nos deux propositions: SUPRA et la Preuve d'Interaction.

Avec SUPRA, nous avons travailé sur la paradigme publish/subscribe utilisé notament par certaines applications IoT pour réduire le nombre de message envoyé dans la blockchain et présenter des garanties de livraison des données. La réduction du nombre de message sur la blockchain permet de réduire le coût des communications et améliorer leurs performances.

Avec la Preuve d'Interaction, nous avons crée un nouveau algorithme de consensus qui ne se base pas sur la puissance de calcul. Ainsi, les noeuds blockchain n'ont plus besoin d'investir dans des ressources de calculs pour gagner un avantage sur la création du bloc. Grâce à cet algorithme, on peut réduire la consommation énergetique des réseaux blockchain et envisager l'ajout d'objet IoT directement dans le réseau blockchain, si on laisse de coté les ressources de stockage nécessaires pour être un noeud.

Futurs travaux

Nous avons des futurs travaux pour nos deux propositions. Pour SUPRA, une idée d'extension est de rendre le protocole compatible avec les systèmes *publish/subscribe content-based*. Il existe deux grandes familles de protocole *pubish/subscribe*, les *topics-based* et les *content-based*. SUPRA fait partie de la première catégorie, mais si on le change directement les *topics* de SUPRA par des conditions sur le contenu de la donnée, SUPRA forcerait les utilisateurs à envoyer plusieurs messages dans la blockchain pour partager la même donnée, un scénario que l'on veut éviter. En appliquant certaines modifications, il serait possble de le rendre compatible avec des systèmes *content-based*. Une deuxième idée d'extension de SUPRA est de retirer une hypothèse sur la taille des transactions de la blockchain utilisée. On suppose que les utilisateurs de SUPRA utilisent une blockchain ayant des transactions capable d'encapsuler entièrement un message SUPRA. Ceci implique que la donnée partagée est directement ajoutée dans le registre, peu importe sa taille. En pratique, plus un message est volumineux et plus les frais de transactions sont élevés. Pour réduire ce problème, l'idée serait de ne plus partager la donnée dans le message SUPRA, mais l'empreinte de cette donnée. Ainsi, le message serait de taille fixe. Le problème est alors de prouver sur la blokchain que la donnée est disponible dans un espace de stockage en dehors de la blockchain, car les smart-contracts ne peuvent que vérifier des informations présentes dans le registre.

Pour la Preuve d'Interaction, notre premier objectif est de retirer notre supposition sur la taille fixe du réseau. En retirant cette hypothèse, nous ferrons un premier pas vers un autre futur travail: la réalisaiton d'une implémentation blockchain utilisant le protocole. Grâce à une implémentation, il serait possible de faire des mesures sur les performances du protocole et de le comparer avec d'autres algorithme de consensus.

Contents

List of Figures

List of Tables

Introduction 1

In 2008, Satoshi Nakamoto presented Bitcoin, a distributed ledger technology [1]. An application where a network of nodes maintains an append-only database without a central authority. Nakamoto's motivation was at that time to create a digital currency that does not rely on banks. The author presented how a network of nodes can maintain identical copies of a ledger, meaning the same list of validated transactions in the same order. The network has an important property: nodes can join or leave whenever they want. It means that the application is completely decentralized, and so not owned by a central authority. Also, because of reasons presented later in this manuscript, data in the ledger is immutable. Once added to the ledger, it is impossible to change a transaction, under certain assumptions that we will also present later. There are different kinds of distributed ledgers, but we will focus in this manuscript on blockchains, distributed ledgers where new transactions are added by block and where each block is linked to the previous block. Bitcoin is one such distributed ledger.

After the publication of this paper, the Bitcoin crypto-currencies was live, which is the first-ever distributed ledger application. This application, and crypto-currencies in general, are just a specific use case for distributed ledgers. In the general case, distributed ledgers allow nodes from the same network to eventually agree on a state, a list of values. In the late '90s, Nick Szabo presented the smart-contracts [6], representations of human contracts into programs. To work, these programs need a trustable execution environment so that all users agree on the current value for the programs' variables. The blockchain can be used as one of such execution environments. Transactions can represent the source code of a program and function calls. By having the same order of transactions, the blockchain nodes will have the same execution for the program. In 2015, Ethereum, the first blockchain implementing smart-contracts was live [7]. Smart-contracts change how to generate trust between users in distributed applications because they allow users to create a program specific for each use case where all users can verify the state of the program.

[6]: Szabo (1998), 'Secure property titles with owner authority'

[7]: Buterin et al. (2013), 'Ethereum white paper'

Blockchain and smart-contracts are used for different kinds of distributed applications but, in this thesis, we will focus Internet of Things applications. Applications where small devices catch information from their local environment and share this data over the Internet or in another communication network. Our purpose is to use blockchains to realistically increase security in IoT. To do so, we first present in Chapter 2 a history of blockchains and explain in further detail how they work. After that, we present in Chapter 3 off-chain solutions, propositions increasing the throughput of blockchain networks, which can be interesting to support the large amount of data generated by IoT applications. Finally, we observe how IoT applications use blockchains in Chapter 4. In our observations, we will see the benefits of the blockchain in IoT, but also two recurring issues:

- some propositions are unrealistic because they do not take into consideration the requirement for using the blockchain.
- some other propositions do not need the blockchain and could use an ordinary database instead.

Based on these observations, we choose to focus ourselves on one specific aspect of IoT, the publish/subscribe paradigm. It is a communication model where a broker acts as an intermediaries between publishers who create events and subscribers who are interested in specific events. This paradigm offers a loose coupling between the publishers and the subscribers [2]. The publishers and the subscribers do not need to know each other to share information, which makes this communication paradigm interesting for IoT or large scale communications in general. In Chapter 5, we present in further detail the properties of this communication paradigm and solutions using the blockchain.

[2]: Eugster et al. (2003), 'The Many Faces of Publish/Subscribe'

1.1 Contributions

In this thesis, we have two main contributions The first one is a publish/-subscribe protocol using the blockchain named Secured Update Protocol with Righterous Accusations (SUPRA). After presenting the publish/-subscribe paradigm in Chapter 5, we observe that using a blockchain can increase the security of publish/subscribe systems, by ensuring data delivery and adding traceability, but the current state-of-the-art solutions have two majors issue: they extensively use the blockchain or they are specific to one blockchain implementation. The extensive usage of the blockchain creates delays for published data and also increases the cost of communications because users have to pay fees to add information to the ledger. On the other hand, having a protocol specific to one blockchain implementation reduces the scenarios where it can be used. With our first contribution, we correct these two problems. We reduce as much as possible the usage of the blockchain and we use generic blockchain functions. We define the environment using the protocol in Chapter 7 and the first version of this protocol in Chapter 8. After that,in Chapter 9 and Chapter 10, we present two extensions for this work: one to correct a scalability problem and the other to add a payment system. Using the blockchain has a cost for users, because they have to pay fees to add data in the ledger. Adding a payment system creates an incentive for using the protocol. In Chapter 6, we present data payment protocols using the blockchain and we will inspire ourselves from these works to define this second extension.

Our second contribution is the Proof of Interaction (PoI), a consensus algorithm. This contribution is presented in Chapter 11. In the Bitcoin Paper, Satoshi Nakamoto presented the first consensus algorithm for blockchain, the Proof of Work (PoW). This specific algorithm is presented in more detail in Chapter 2. Because of reasons also explained in this chapter, nodes invest in computational resources to increase their chance of profit. The result is that today, the energy consumption of the Bitcoin network is comparable to a country [*]. With our second contribution,

[*] As of July 2022, the energy consumption of the Bitcoin network is equal to the energy consumption of Argentina
https://digiconomist.net/bitcoin-energy-consumption/

the PoI, we present a consensus algorithm that does not require nodes to invest in computational resources. In our proposition, nodes have to sequentially interact with other nodes in the network to add a new block. This consensus algorithm reduces the energy consumption of blockchain networks and also reduces the risk from a well-known attack on PoW systems: selfish mining.

1.2 List of publications

The propositions presented in this manuscript were published during the thesis in several papers. In Table 1.1 are listed all the conferences and journals in which these works are published.

Table 1.1: List of publications

Authors	Title	Conference
Jean-Philippe ABEGG Quentin BRAMAS Thomas NOEL	Comment gagner de l'argent sans travailler ?	ALGOTEL 2020
Jean-Philippe ABEGG Quentin BRAMAS Timothée BRUGIERE Thomas NOEL	SUPRA, a distributed publish/subscribe protocol with blockchain as a conflict resolver	BRAINS 2021
Jean-Philippe ABEGG Quentin BRAMAS Thomas NOEL	Blockchain using Proof-of-Interaction	NETYS 2021 (best paper award)
Jean-Philippe ABEGG Quentin BRAMAS Timothée BRUGIERE Thomas NOEL	Distributed Publish/Subscribe Protocol with Minimum Number of Encryption	ICDCN 2022
Jean-Philippe ABEGG Quentin BRAMAS Thomas NOEL	SUPRA un protocole publish/subscribe distribué	ALGOTEL 2022
Jean-Philippe ABEGG Quentin BRAMAS Thomas NOEL	Secret-less secured payment system for inter-broker communications	WiOpt 2022

Authors	Title	Journal
Jean-Philippe ABEGG Quentin BRAMAS Timothée BRUGIERE Thomas NOEL	SUPRA, a distributed publish/subscribe protocol with blockchain as a conflict resolver	PeerJ (in submission)

State of the art

Blockchain 2

In this section, we explain how runs a blockchain and what are the current challenges with this technology.

2.1 General concepts

Blockchain is a distributed ledger technology presented in 2008 in the Bitcoin Paper by Satoshi Nakamoto [1]. Distributed Ledger Technologies (DLT) are data storage methods that create an append-only database in a network of untrusted nodes [8]. Each node is controlled by an individual or an organization and stores a full copy or a part of the ledger, which makes the ledger highly available. The lack of trust between the nodes is the main difficulty to handle because there is no central authority to update the ledger. To resolve this trust issue and have the same copy of the ledger (*i.e.* same transactions in the same order) among the nodes , the nodes run a consensus algorithm. For blockchains, this consensus algorithm is a leader election.

[1]: Nakamoto (2008), *Bitcoin: A Peer-to-Peer Electronic Cash System*

[8]: (2020), 'Trade-offs between Distributed Ledger Technology Characteristics'

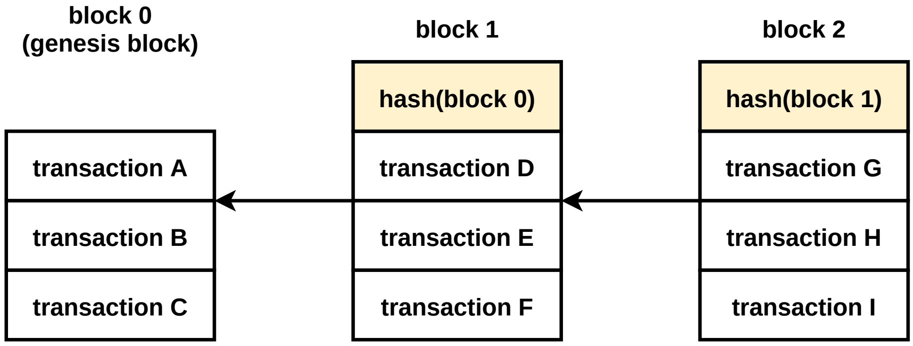


Figure 2.1: Chain of blocks

The ledger contains transactions and, in the case of blockchains, these transactions are stored in blocks, where each block has a reference to the previous block. It creates a chain of blocks, hence the name blockchain. In Figure 2.1, we can observe a representation of this notion of chains of blocks. Also, we can notice in the figure that the first block, sometimes named the *genesis block* in the literature, is the only block with no pointer to a previous block.

With this chaining, nodes can detect unwanted updates on the ledger. Indeed, if a transaction is modified in block i, it will change the fingerprint of block i, and so the fingerprint of block i+1 since the pointer to the block i is stored in i+1. So these unwanted updates in block i will have a chain reaction until block n, the last known block by the nodes. Combined with the consensus algorithm that only selects one node to add a new block, once a transaction is added to the ledger, the probability that a modification can be made decreases exponentially fast over time. That is why data on the blockchain is considered immutable. We explain in Section 2.2 under which assumptions this property remains true.

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 nodes verify the signature using the author's public key. Some transactions can have multiple authors, in that case, the signature of all the authors is required to consider the transaction valid.

Once a node receives a transaction, it broadcasts it over the network. The whole network will eventually be aware of this transaction. 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 challenge of DLT is for the nodes to keep the same copy of the ledger, which means having the same transactions and applying them in the same order among all nodes, without a central authority. To do so, the nodes must execute a consensus algorithm between them.

2.2 Consensus

2.2.1 Consensus problem

The consensus problem is a well known problem in distributed computing. The most famous presentation of this problem is The Byzantine Generals Problem by Lamport [9]. In this example, a set of generals besieges a city and they must decide if they will attack or not tomorrow. Generals cannot talk directly with one another. Each one stays in a camp and they send messengers to exchange information. The trick is that some Generals are loyal to the cause and some other are traitors, they are named Byzantines.

[9]: Lamport et al. (1982), 'The Byzantine Generals Problem'

A special General, the Commander, share an order with all the other Generals, called Lieutenants. While a loyal General will share the same order with all the other generals, a Byzantine General can lie and share contradictory orders to the other Generals. The purpose of the problem is for the loyal Generals to agree on the order from the Commander and applying it. To resolve this problem, the Generals must reach a consensus.

We can easily make the analogy between this problem and computer science: the Generals are nodes and the Byzantine Generals are faulty or malicious nodes, the messengers are messages exchanged over the network, and the decision is just a value for a data. In blockchain, the consensus is used to ensure that all nodes have the same new block. This means, that all nodes select the same transactions to update their ledger with and order them the same way. So each time nodes reach a consensus, they agree on the same new block, so nodes maintain the same copy of the ledger.

Formally, if we want to achieve the consensus between nodes with correct nodes (*e.g.* nodes following the protocol) and faulty nodes (either crashing or Byzantine nodes), these 2 guarantees must be verified:

- Liveness: all correct nodes eventually decide on a value.
- Safety: all correct nodes that decide should decide on the same value. If all nodes are correct and have the same initial input, that value should be the only possible decision value.

There exist algorithms verifying these properties in different kind of models. The models differ on the message format and the network delay. With the usage of a Public Key Infrastructure (PKI), using a communication model where messages are signed by the authors and cannot be tampered with is common. Lastly, there are 3 communication models in which is considered the consensus:

- Synchronous model: messages will reach the destination before a known delay Δ.
- Partially synchronous model: after an unknown global stabilization time, messages will reach the destination before a known delay Δ.
- Asynchronous model: message delivery delay is not bounded.

In these models, there exist several propositions. For instance, Dolev et all. [10] presented a consensus algorithm that works no matter the number of Byzantine nodes in the system in a synchronous model. On the other hand, in a partially synchronous model, it has been proved that the consensus cannot be reached if more than $1/3$ of the nodes are Byzantine [11]. Lastly, in an asynchronous model, it has been proved that the consensus cannot be reached if just one node is Byzantine [12].

As we can see, the consensus problem is a well-known problem already solved or proved unsolvable in several types of models. From this observation, we can ask ourselves what is the novelty introduced by Satoshi Nakamoto? All the examples mentioned above share a common assumption: the number of nodes is known in the network. It is what we called a permissioned network. This type of network makes sense to solve consensus among a fixed set of participants, for instance, in distributed databases. The novelty of the blockchain presented by Nakamoto is to solve the consensus problem in a dynamic network where nodes can join or leave at any time during the election process, and where nodes do not know the total number of nodes in the network. This is what we called a permissionless network.

2.2.2 Permissionless consensus

In permissioned environments, since you know the number of nodes in the system, the consensus algorithms used in this setup can be summed up as one node equals one vote. The consensus can be solved with a voting system where the most represented decision among honest nodes will be the decision taken by all the honest nodes. Unfortunately, this kind of system cannot be applied in a permissionless network for two reasons. First, nodes do not know the number of nodes, and so the number of votes necessary to have a winning majority. Second, since nodes can join or leave at any time, a voting system could not handle a Sybil attack [13]. In this kind of attack, a malicious entity generates several new identities in the network to increase its voting power. For the rest of the nodes, the votes came from several users and may be legit, while in fact, they all came from the same user.

[10]: Dolev et al. (1983), 'Authenticated algorithms for Byzantine agreement'

[11]: Dwork et al. (1988), 'Consensus in the presence of partial synchrony'

[12]: Fischer et al. (1985), 'Impossibility of distributed consensus with one faulty process'

[13]: Douceur (2002), 'The sybil attack'

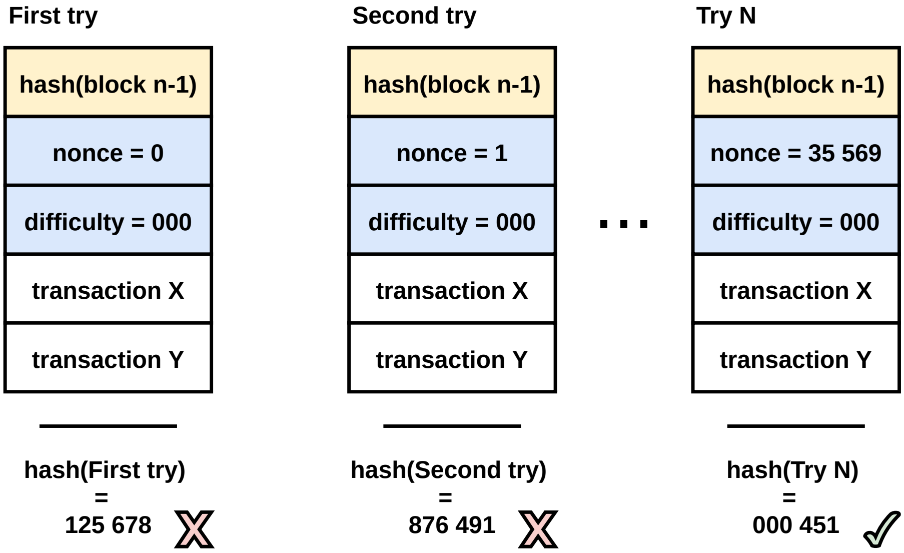


Figure 2.2: Simplify representation of the Proof of Work to propose the block n

From a denial of service protection technique [14], Satashi Nakamoto finds a solution to both issues and presents an algorithm that resolves the consensus problem in a permissionless environment, in a synchronous model, called the Proof of Work (PoW) [1]. In this algorithm, to be selected as the leader, a node needs to answer a problem: finding a correct fingerprint for a possible new block.

[14]: Back (2002), 'Hashcash - A Denial of Service Counter-Measure'

In Figure 2.2, we have a representation of the proof-of-work from a node's point of view. To be allowed to propose a new block, a node must find a value for a field of the block called the nonce such that the fingerprint of the block is smaller than the target value. The target depends on the difficulty. The higher the difficulty, the smaller the target. Fingerprints cannot be predicted, one must compute them to know if they are correct or not. From this property, there is only one strategy for the nodes: trying values as fast as possible.

If a node finds a correct nonce, it broadcasts its block in the network. The other nodes check if the nonce is correct and, if the nonce and all the transactions inside the block are correct, add the block to their local copy of the ledger. The node that finds a new block receives a reward, and the network starts searching for a new block. This reward system encourages the nodes to stay in the network and keep alive the ledger. The reward for finding the block is the sum of all transaction fees inside the block if transactions have fees, and a flat bonus.

The computational power of the node is linked with its chance to find the new block, because the more values a node can try, the more chance it has to find a correct nonce before the other nodes. Using this resource is what allows this consensus algorithm to resolve the absence of knowledge on the number of nodes in the network, because all the computations made by the node are done locally, without caring about the other nodes. Also, computational power is what handles Sybil attacks [13]. In this kind of attack, a malicious entity increases its voting power by registering itself several time in the network with different identities. With the PoW, to increase the chance of finding a new block, an entity must invest in computational resources, because the more values you can try the more chances you have to find a correct nonce. This means that this entity can be registered under one or N identities in the network, its computational power will remain the same, and so its voting power.

The difficulty that the block must match with its proposition represents roughly the number of 0 in front of the fingerprint. This value is periodically updated to take into account the computing power of the network. In the case of Bitcoin, the first-ever blockchain, the difficulty is readjusted every 2 weeks so finding a new block takes 10 minutes on average for the current computational power of the network.

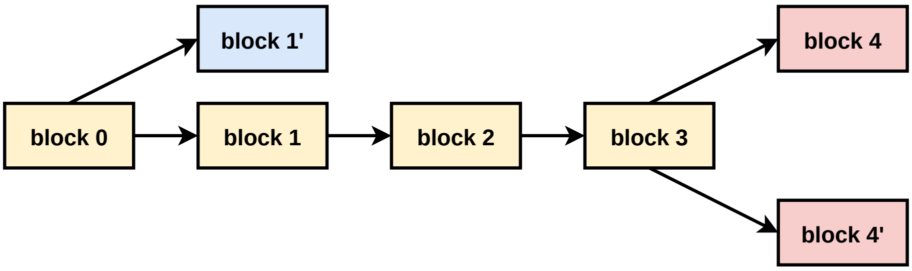


Figure 2.3: Chain of blocks with a fork on the last block

It is possible for the chain of blocks to fork periodically. Meaning, that there are two, or more, valid propositions for the new block. This kind of event is represented in Figure 2.3. We can observe on the figure that for the last block, block 4, there are two possible blocks. The origin of this event is linked with how the network is notified by a new block. Nodes broadcast to the rest of the network new blocks when they are aware of their existence. This notification can take time to reach the whole network and, during this delay, another valid proposition for the current block can be found in an unnotified part of the network. The result is that the network has two valid propositions and must find which one will remain in the ledger. We can notice that these two possible blocks may have exactly the same transactions in them, but they are proposed by different nodes, and so reward different nodes.

To know which proposition will remain in the ledger, nodes follow one specific rule: the branch with the biggest total difficulty will be selected. Each node will choose a proposition as the previous block and try to find a new block, which will split the computational power of the network between the two branches. The proposition linked by the next block will be the one added to the ledger, making it the main branch. In case of tie with the total difficulty, the first known block will be selected by the node. Pass et all.[15] proved that we need a known upper bound on message delay to ensure the consensus in a PoW system. For forks, it means that the honest nodes eventually chose the same branch between all forks for the ledger. It also means that, in a PoW system, a block B is not considered immutable once it is added to the ledger but once enough new blocks where found after B so that no forks can change B.

With the PoW, it is necessary that at least $1/2$ or more of the total computational power of the network has to be owned by honest users. Otherwise we cannot guarantee that data is immutable, so the integrity of the ledger can be attacked. Let's say that a Byzantine changes a transaction in a block i. This modification will change the fingerprints of all the

[15]: Pass et al. (2017), 'Analysis of the blockchain protocol in asynchronous networks'

blocks from i to the last know block j, so it invalidates the nonces in them. The Byzantine must recompute all the nonces, but once it recomputes all the nonces, j is no longer the last known block. Indeed, honest nodes kept working and have now found k as the last known block. The Byzantine must now also computes the nonces for the blocks from j to k if it wants its modification to be considered in the main branch. If it owns more than $1/2$ of the total computational power of the network, the Byzantine node will catch up with the main branch and so modify a transaction in an old block. They can rewrite the ledger.

Still, by using the fork mechanism, it is possible to attack the PoW ledger with a computational power between $1/3$ and $1/2$ of the total computational power with an attack named selfish mining. This attack can increase the profit of the attacker and allow it to choose the transactions in the ledger. It can be done by a malicious node, or a set of malicious nodes, finding the next valid block i. Instead of sharing the block, the malicious node keeps the block for itself and begins the PoW for the block $i+1$. When a correct node finds a proposition for block i, the selfish node shares its stored proposition for block i, creating a fork. Based on the rule previously explained, the network will split itself between the two valid propositions for block i, and the block $i+1$ will decide which version of block i will remain in the ledger. While the selfish node stored its valid proposition for block i, the network wastes energy because it was computing the PoW of a block already found and while the malicious node tried some values for the block $i+1$'s nonce. Meaning that its proposition for block $i+1$ has more chance to be added to the ledger, and so the node has more chance to win the reward. These attacks increase the transaction validation time and allow malicious nodes to increase their profit [16].

[16]: Tschorsch et al. (2016), 'Bitcoin and beyond: A technical survey on decentralized digital currencies'

After the PoW, S. King and S. Nadal [17] proposed an alternative called the Proof of Stake (PoS). In this consensus algorithm, each node taking part in the consensus locks tokens. Then a node with locked funds is randomly selected to propose the next block. The biggest the stack, the higher the chance of being selected. To prevent the node with the biggest number of tokens from always being selected, each token has an age. The stake of a node is the sum of the age of all its locked tokens. With each block token's age increases, and so does the node's stake. Once a node has been selected and has proposed a new block, the age of all its locked tokens is set to 0. This election mechanism requires less computational power but has security issues [18, 19] (*e.g.*, Long-range attack and DoS). Still, by resolving the consensus in a permissionless environment, the blockchain opens new applications.

[17]: King et al. (2012), 'Ppcoin: Peer-to-peer crypto-currency with proof-of-stake'

[18]: Gaži et al. (2018), 'Stake-bleeding attacks on proof-of-stake blockchains'

[19]: Bonnet et al. (2020), 'Stateless Distributed Ledgers'

2.3 Applications

In the previous sections, we explained that a distributed ledger contains signed transactions and that nodes use a consensus algorithm to add new content to the ledger, we focused on a specific type of ledger: blockchains. In this section, we explain in which case distributed ledger technologies are interesting.

When Satoshi Nakamoto published the Bitcoin paper, his purpose was to create a decentralized payment system without a trusted third party. This means a payment system that does not rely on banks. In 2009, he reaches his objective, and the first-ever blockchain application was live: Bitcoin crypto-currencies.

A crypto-currency is an application using a distributed ledger technology to create a peer-to-peer payment system. They can use a blockchain, like Bitcoin, or some use other kinds of distributed ledgers. For instance, Iota [20] uses a distributed ledger-based on a directed acyclic graph, the Tangle. The distributed ledger is here to prevent double-spending attacks. This means preventing a malicious user from spending two times the same coins to pay two different users. Since a transaction cannot be modified once added to a block, this type of attack is impossible with blockchains, as long as more than $1/2$ of the computational power of the network remains honest. Also, since users are identified by a public key, and not a name, they are anonymous while usaly remaining traceable.

[20]: Silvano et al. (2020), 'Iota Tangle: A cryptocurrency to communicate Internet-of-Things data'

Crypto-currencies are the first-ever blockchain applications, and the most famous ones, but they are only a specific use case. In fact, based on the content allowed in the transaction, distributed ledger technology can be used in several other use cases. Each distributed ledger protocol has its own rules on the content of the transactions. The nodes are aware of these rules when they join the network and they check the validity of each transaction inside the block to know if the block is valid, on top of the consensus algorithm.

In the general case, transactions inside a distributed ledger can contain any type of data. If transactions can contain generic data, the ledger became some kind of decentralized non-repudiation database. Since transactions are signed by their author, once information is added to the ledger, its author cannot deny the existence of the event. This usage is interesting but limited.

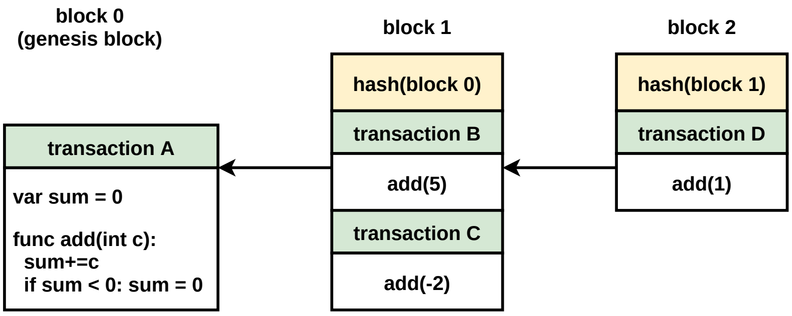


Figure 2.4: Declaration of a simple smart contract, and calls to its function. Function calls are executed in the order indicated in the blocks. The final value of sum is 4 in this example.

In the nineties, Nick Szabo [6] presents a technology called smart contracts. The objective was to translate human contracts into programs having secured and trustable executions. These properties needed by the smart contract can be given by a distributed ledger, because the nodes can verify the correct execution of the contract and the consensus ensure that all the nodes have the same execution. Based on this observation, Ethereum [7] was launched as the first blockchain that includes a programming language allowing users to create smart contracts.

[6]: Szabo (1998), 'Secure property titles with owner authority'

[7]: Buterin et al. (2013), 'Ethereum white paper'

In Figure 2.4, we can observe how a smart contract can run in a blockchain. In the example, a smart contract is declared in the genesis block. This contract has only one function, which increments a variable stored by the contract, and verifies that the variable is always positive or nill. In the next blocks, the function is called three times. With this simple example, we can observe that blockchains allow global ordering of function calls between the nodes. This is very important, because in our example, with three function calls, the final value of sum can be 4 or 6, based on how the transactions are ordered in the second block. The nodes execute function call like they are ordered in the blocks from the genesis block to the last known block. This allows nodes to share a global state for the variables. Honest nodes will eventually have the same states, which makes the program execution secured. This example is very simple, but we will see later smart contracts that execute more complex tasks.

Notice that once a smart contract is added to the ledger, since the transaction cannot be modified, the contract cannot change. This is why it is important to check the code of the contract before deploying it. Also, based on the complexity of the function called in the transaction, the user has to pay fees.

To summarize, blockchains are trustable and transparent decentralized execution environments. This environment was at first used to generate new electronic payment systems and was then used to run smart contracts, which opens new possibilities to generate trust among users in distributed environments.

2.4 Challenges

In this section, we present some of the current known challenges in distributed ledgers.

2.4.1 Resources usage

Being a node uses resources: computational resources to verify blocks, transactions, and executing the consensus algorithm, but also storage resources to store the ledger.

For the computational resource, the biggest concerns are from the Proof of Work (PoW). Indeed, this consensus algorithm is a computational race between nodes. The first node that finds the next block will get a reward. This incentive encourages nodes to invest in computational resources because the higher the computational power, the more chance you have of finding a next bloc. This investment in computational resources increases the energy consumption of Blockchain networks using PoW. Today, the energy consumption of these networks is equal to the energy consumption of developed nations *. Which is a concern for environmental questions.

We talked about the Proof of Stake (PoS), which reduces the computational power needed to execute a consensus, but this proposition has security

* As of July 2022, the energy consumption of the Bitcoin network is equal to the energy consumption of Argentina
https://digiconomist.net/bitcoin-energy-consumption/

issues. Intel proposed another alternative to the PoW, the Proof-of-Elapsed-Time (PoET) [21]. This solution requires Intel SGX as a trusted execution environment. Thus, Intel becomes a required trusted party to make the consensus work, which might imply security concerns [22] and is against the idea of removing third parties. Other mechanisms were proposed such as Proof-of-Activity and Proof-of-Importance [23], which are hybrid protocols or protocols using properties from the network itself. To recap, currently, there is no consensus algorithm for a permissionless network having the same security properties as the PoW without using as many computational resources.

The second issue with the resource consumption of the blockchains is the usage of storage resources. The blockchain is an append-only ledger. This means that the size of the ledger just grows overtime, since no transaction or block can be deleted. This means that the nodes will have to invest indefinitely in storage resources. To resolve this problem, we find two types of propositions in the literature: locally change the ledger or globally change the ledger.

By reducing locally the size of the blockchain, a node can lose the ability to verify new blocks and transactions and be part of the consensus or check the validity of old blocks and transactions. In this case, we can call this node a Light Node. By opposition with Full nodes, nodes storing a full copy of the ledger. Light nodes need to be connected to a trusted full node to get the ledger's last update and verify information not stored by the light node.

To reduce locally the size of the ledger, each node chooses local parameters. These parameters can be a maximum amount of complete stored blocks [24], or a subset of accounts that the light node wants to follow [25]. Based on these parameters, the light node executes local changes to the ledger. This is a one-way transformation, if the node wants to get back the full ledger, it must download it from a full node.

In fact, reducing locally the size of the blockchain does not resolve the storage resource consumption problem, because the ledger's size still grows on the full nodes, the nodes taking part in the consensus and keeping alive the ledger. These nodes will eventually reach their storage limit, and when all nodes have reached it, no more blocks will be added. That is why, we need to reduce globally the size of the ledger.

Several works propose methods to reduce globally the size of the blockchain. In these propositions, some introduce a maximal size for the blockchain [26–28], while others reduce the blockchain's growth by changing some fields in the block or the transaction format [29–32]. Some propositions present how to create a new genesis block in the ledger that sum up all the balances in the ledger [33, 34].

The difficulty in this exercise is to reduce the size of the blockchain, without reducing its security. Users must have guarantees that they can still retrieve the current state of the ledger: the balance on each account or the variables inside smart contracts. Otherwise, users will lose trust in the protocol, and the ledger will be useless. There is a lot of proposition to spare storage resources but, in the most popular blockchains, like Bitcoin or Ethereum, none of them are implemented.

[21]: (), *PoET 1.0 Specification*

[22]: Chen et al. (), 'On Security Analysis of Proof-of-Elapsed-Time (PoET)'

[23]: Alsunaidi et al. (2019), 'A survey of consensus algorithms for blockchain technology'

[24]: Palai et al. (2018), 'Empowering Light Nodes in Blockchains with Block Summarization'

[25]: Palm et al. (2018), 'Selective blockchain transaction pruning and state derivability'

[26]: Matzutt et al. (2020), 'How to Securely Prune Bitcoin's Blockchain'

[27]: Matzutt et al. (2021), 'CoinPrune: Shrinking Bitcoin's Blockchain Retrospectively'

[28]: Kim et al. (2019), 'SCC: Storage Compression Consensus for Blockchain in Lightweight IoT Network'

[29]: (2018), 'Recycling Smart Contracts: Compression of the Ethereum Blockchain'

[30]: Chen et al. (2019), 'Bitcoin Blockchain Compression Algorithm for Blank Node Synchronization'

[31]: Zheng et al. (2019), 'An Innovative IPFS-Based Storage Model for Blockchain'

[32]: Kumar et al. (2019), 'Implementation of Distributed File Storage and Access Framework using IPFS and Blockchain'

[33]: Sliwinski et al. (), 'Asynchronous Proof-of-Stake'

[34]: Sliwinski et al. (2019), 'ABC: Proof-of-Stake without Consensus'

2.4.2 Transaction rate

The nodes follow a blockchain protocol that they trust. This protocol contains the blocks and transaction formats, and the consensus algorithm that the nodes must execute. Since blocks have a maximal size, and the consensus algorithm ensures that new blocks are added at a constant rate, on average, the blockchain network has a maximal rate of transactions added to the ledger. This rate cannot be changed, unless we update the protocol. For instance, in Bitcoin, the network adds, on average, 6 transactions per second with one block every 10 minutes.

This transaction rate is shared between all the applications using the blockchain, and the more the blockchain become popular the more users want to add transactions to the ledger. Unfortunately, since the maximal rate will not change, blockchains have a scaling issue between the transaction rate and the popularity of the network.

To resolve this issue, there are several propositions. For instance, changing the block format to have more transactions in the block. That is what Bitcoin Cash does. It is a blockchain protocol based on Bitcoin with a different block format that allows more transactions. Another solution is to use a faster consensus algorithm. Consensus algorithms used in permisionless network are slow. Since we do not know the number of nodes, creating new blocks takes time to ensure that the maximum number of nodes are aware of the last block before the creation of a new one. By using a permisionned network, the nodes can use consensus algorithm based on vote like PBFT [35]. This kind of consensus algorithm is faster than permissionless consensus algorithms, but they remove one interesting property of the blockchain network.

In fact, changing the block format or the consensus algorithm does not resolve the transaction rate problem, it only postpone it. It will take more time for the blockchain network to be overwhelmed by transactions but, if the popularity of the blockchain network keeps increasing, the maximal transaction rate will be reached. In the next section, we will present what we call off-chain solutions, or Layer-2 solutions. Solutions that allow the blockchain users to securely exchange transactions without adding them in blocks.

[35]: Castro et al. (1999), 'Practical byzantine fault tolerance'

Off-chain solutions | 3

Nodes can only add a limited number of transactions in each block and the delay between blocks is constant, thus there is a limit on the rate of new transactions that the blockchain network can handle. This limit can prevent the usage of blockchain in large-scale applications. For instance, using cryptocurrencies as legal tenders in a country is impossible, because the blockchain network as it is will not be fast enough for the population's everyday transactions.

In this section, we discuss off-chain solutions, solutions that resolve the scaling problem of the blockchain. These kinds of propositions are sometimes referred to as Layer-2 protocols because they work under Layer-1 protocols: the distributed ledgers. We will present in detail two solutions: the commit-chains, and the state channels.

3.1 Commit-chains

3.1.1 General idea

Commit-chains are introduced in 2017 by Poon et al [36] in the Plasma white paper. The idea is to create nested blockchains, each handled by a designated user called the operator, and secure these chains with smart-contracts [37]. The blockchain on which is added the smart contract is called the parent chain, and a chain managed by an operator is a commit-chain. By creating nested chains, the parent chain virtually increases its transaction rate, because while the parent chain manages transaction as explained in the previous section, the commit-chains manages the transactions between a subset of users. Each commit-chain has a parent chain and this parent chain can also be a commit-chain. The parent chain with no parent is called the root chain. It creates a tree of chains like in Figure 3.1.

[36]: Poon et al. (2017), 'Plasma: Scalable Autonomous Smart Contracts'

[37]: Khalil et al. (2018), 'Commit-Chains : Secure , Scalable Off-Chain Payments'

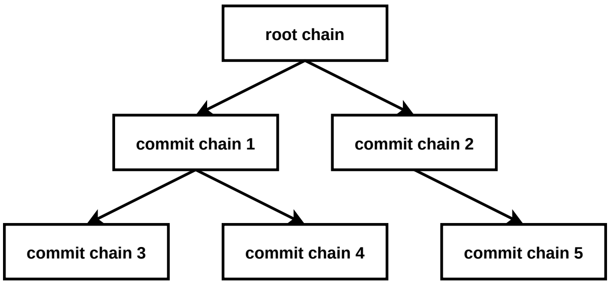


Figure 3.1: Each node represents a whole chain. The nest of commit-chains forms a tree.

Each commit-chain is managed by a smart-contract deployed in the parent chain. To join the commit-chain, users from the parent chain register them-selves by calling a function of the smart-contract. Once they are registered, they can deposit tokens in the commit-chains. After that, users registered in the commit-chains can exchange tokens between them without sending transactions to a node from the parent chain. Instead, they broadcast the transaction in the commit-chains network indicated in the smart-contract. The transaction is added to a block of the commit-chain. Periodically, the operator has to indicate to the parent chain the current state of the commit-chains. For instance, on Figure 3.2, we can observe that the operators report new blocks at the same rate as the parent chain. The period between each state report in the parent chain, and the consensus used to create new blocks in the commit-chains are defined in the smart-contract.

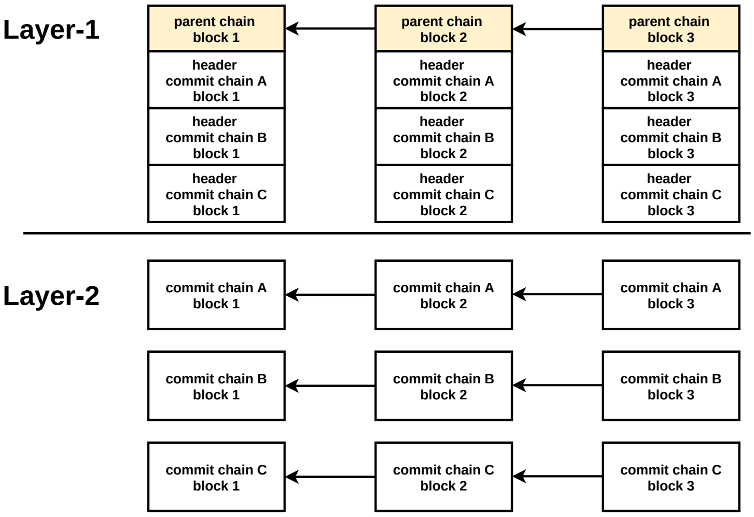


Figure 3.2: A blockchain with three commit-chains

The operator looks like a trusted third party in the commit-chain, because it is the only entity that updates the state of the commit-chain, but the operator can be malicious. Malicious operators cannot cheat on honest commit-chain users, thanks to the smart-contract, as explained next.

3.1.2 Smart-contract

The smart-contract ensures security in the commit-chain, because it defines the rules of the commit-chain. Commit-chain users must verify the smart-contract before joining the chain. The contract describes how the blocks are added to the commit-chains, how often the operator has to share the state of the commit-chain to the parent chain, and how conflicts are resolved.

Describing how to add blocks means defining the consensus algorithm used in the commit-chains. In this case, we will name validators the entities being part of the consensus algorithm. Notice that it is possible that not all the commit-chains users are validators or even that there is only one validator. The commit-chains network is a permisioned network. Since members have to register themselves in the contract, it is possible to use the consensus algorithm faster than the Proof of Work between the validators. When new blocks are made, they are broadcasted to the members of the commit-chain, and not added to the parent chain.

The operator has to periodically report to the parent chain the state of the commit-chain. This periodicity is indicated in the smart-contract. If the periodicity is not respected, the operator is penalized by the contract. We observe that this report process works because it assumes that there is an upper bound on the time to add a transaction to the ledger. Otherwise, we cannot ensure that honest operators can report in time to the parent chain.

Reports are used to detect conflicts between the members of the commit-chains. A conflict happens when a member of the commit-chain thinks that the report of the operator is incorrect. To prove the error, an honest user must present a fraud proof. The format of the proof is described in the contract and depends on the consensus algorithm used in the commit-chain. If the smart-contract accepts the proof, the last report is invalidated. The commit-chain switches to the state indicated in the previous report, and the operator is penalized.

The penalty received by a malicious operator are indicated in the smart-contract. They can be different on each commit-chains but, at least, the contract selects a new operator. Otherwise, the commit-chain could be blocked, if the operator decide to not send new reports. Also, conflict resolution can happen during a certain delay after adding the report to the parent chain. When the delay is over, it is not possible anymore to report possible fraud. That is why the members of the commit-chains must periodically check the parent chain to check each new report.

If the operator wants to maximize its revenue, it has no interest in having malicious behaviors since it receives some of the transaction fees in the commit-chain network as payment for its service and it will get penalized if honest users can detect the incorrect behavior.

3.1.3 Pros/Cons

Table 3.1: Pros and cons for commit-chains

Pros	Cons
- commit-chains reduce the number of transactions stored in the root chain.	- users have to periodically check the blockchain.
- commit-chains increase the transaction validation rate of the blockchain network.	- leaving a commit-chain takes time.
- commit-chain transactions have smaller fees.	

We summarize the pros and cons of commit-chains in Table 3.1. As a layer-2 solution, the first advantage of commit-chains is to increase the scalability of the root chain with the number of transactions. Transactions occurring between users of the commit-chain are added to a block in the commit-chain. These transactions will not be added to the root chain, which spares storage resources for the root chain's nodes. Also, the transaction throughput of the whole blockchain network increases. The total throughout became the sum of the throughput of each commit-chain and the root chain. Also, since commit-chains can use permisioned consensus algorithm, transactions can be added more quickly in blocks.

There is also a financial advantage for commit-chain users. The throughput of the blockchain is limited but the more the blockchain has users, the more new transactions are made. Since the number of transactions in a block is limited and the nodes wants to maximize their profit, nodes select the transactions with the most fees in their block. So to be selected in blocks, users increase the fees in transaction, the more there is new transactions. In commit-chains, since there are fewer users, there are fewer new transactions and the transaction fees can be reduced.

It also means that it is possible to deploy more complex smart-contracts. We said earlier that, based on the complexity of the function called in a smart-contract, the user has to pay fees. With commit-chain, it is technically possible to split a complex smart contract into smaller smart-contracts into several commit-chains. Instead of doing the whole computation on a single blockchain, smaller functions in commit-chains are called and the result is shared in the parent chain and used in the smart contract on this chain.

The main issue with commit-chains is the time required for users to unlock its tokens from a commit-chain. To withdraw tokens, a user must indicate to the smart-contract the number of tokens it wants to remove and prove that it has these tokens in the current commit-chain states. When it is done, the smart-contract waits for a delay before transferring the tokens to the users in the parent chain. During this delay, honest users have time to present to the smart-contract a proof that the user leaving the chain is malicious and does not own the tokens. This property can be an issue for users with a small number of tokens who need to use tokens on the root chain. Especially if users have to unlock funds on several layers of commit-chain before reaching the root chain.

Also, the users have to check the validity of the state presented by the operator to the smart-contract, and so users have to periodically check the blockchain. Otherwise, a malicious operator can present an invalid state to the parent chain without receiving a penalty.

3.2 State channels

State channels are another layer-2 protocol used to scale the blockchain transaction validation rate with a large number of incoming transactions. Where commit-chains create smaller blockchain networks to exchange transactions between a subset of users, state channels are focused on communication between two users. Once the state channel is up, these two users can exchange an unlimited amount of transactions, and confirm them instantly, and without paying fees. To present this method, we will explain Lightning, the protocol that introduced the concept of state channels, and we will present how it can be used.

3.2.1 Lightning Network

[38]: Poon et al. (2015), 'The bitcoin lightning network'

Lightning Network was presented by Poon et al [38], the same author as commit-chains. The motivation of this proposition is to make the Bitcoin network as fast as, or even faster than, the Visa payment network. Bitcoin validates 6 transactions per second, whereas Visa had a peak at 47 000 transactions per second in 2013 [38]. To scale up, the idea is to create a Lightning Channel between two users A and B.

Before explaining how the channel is created and used, we must explain in which case it is interesting to open a channel. The Lightning channel aims for situations where two users, we will name them A and B, often

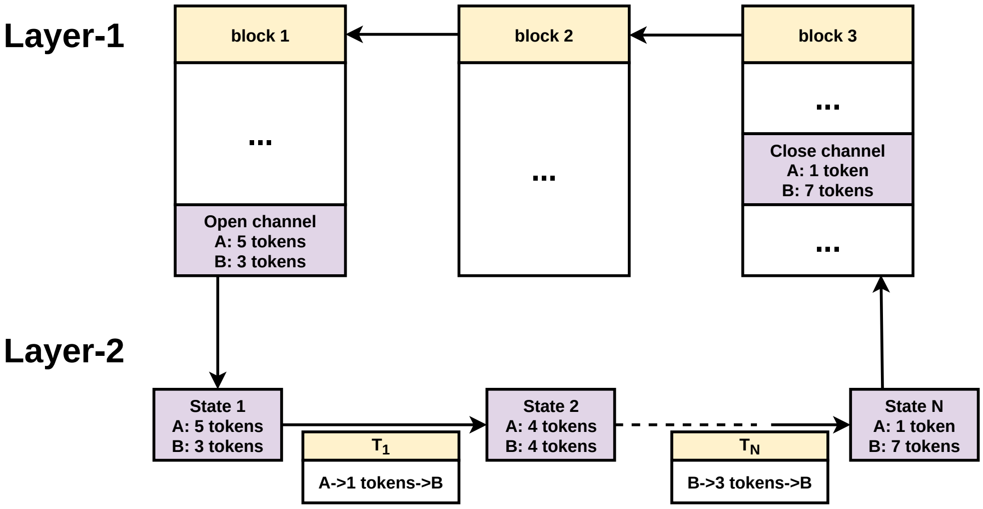


Figure 3.3: Opening and closing of a Lightning channel between A and B.

have to exchange a small number of tokens. For instance, A can be a regular customer of a shop owned by B. With the classic Bitcoin payment system, if A wants to buy something, B has to wait several hours before being sure that the payment is confirmed. On top of this, A has to pay fees. If the payment is regular, the cost of the payment system became unneglieable. We observe that the Bitcoin network is not an interesting solution for micro-payment, the Lightning Network wants to propose a solution to this exact situation. It allows A and B to directly confirm the transactions exchanged between them without paying fees.

In Figure 3.3, we represent the opening, the usage, and the closing of a Lightning channel. To open the channel, A and B must add a transaction in the ledger where they both lock some tokens. This commitment transaction is signed by both users to prove the agreement between the two users. When this transaction is added to the ledger, the channel is open and A and B can exchange an unlimited amount of transactions involving those tokens between them. With each transaction, we create a new state between A and B, meaning a new distribution of the token locked in the commitment transaction. Whenever they want, A and B can stop the channel and leave the system with the last valid distribution of tokens. If both users want to stop the channel, they can sign a final transaction with the last distribution, and, when this transaction is added to a block, the channel is stopped. Otherwise, the channel is stopped after a known delay, to prevent malicious users to leave the system with an old distribution.

How can we detect a malicious user trying to stop the channel with an incorrect token distribution? For instance, a token distribution where the malicious user has more tokens than what it is supposed to have. To make the system secure, the Lightning network uses a Hashed Timelock Contract (HTLC).

3.2.2 Hashed Timelocked Contract

The HTLC is a transaction that has two outcomes, one based on a secret revelation and one based on a timelock. To explain HTLC, we represent an example of usage between two users in Figure 3.4.

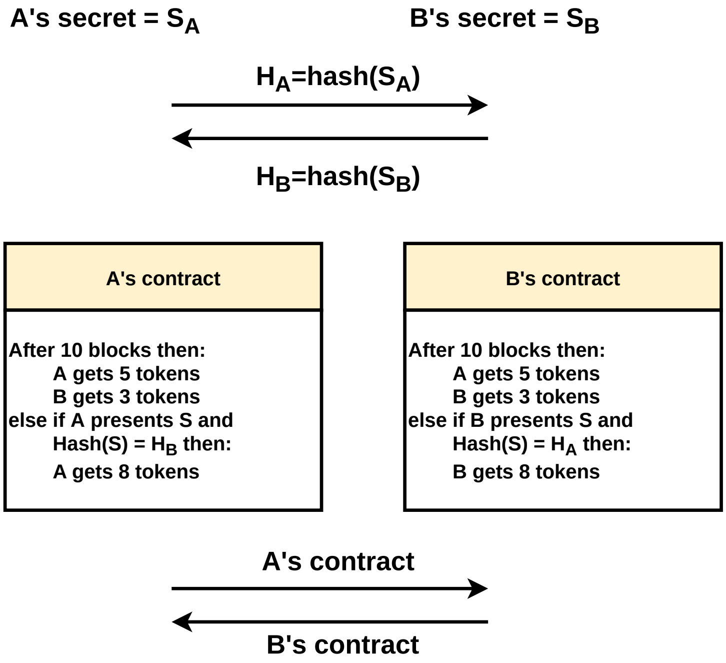


Figure 3.4: Creation of a new state between A and B with a HTLC.

Let's assume that A and B have opened a Lightning channel. When A and B want to exchange tokens, each user generates a secret value and shares the fingerprint of the secret with the other user. With the fingerprint, A and B generate each one a HTLC. In the contract, they put a condition on a delay. If the contract is added to the blockchain, A and B will receive the token distribution indicated in the contract at the end of the delay. Otherwise, during this delay, if the secret is revealed, the other user will obtain all the tokens locked in the channel. A and B then exchange the contract. With this, A's contract own by B became useless for B as soon as S_B is revealed. In other words each transaction is revocable.

If A and B want to exchange again tokens, they have to generate new secrets to generate new contracts and reveal the secrets of the previous contracts. This secret revelation scheme prevents malicious users from stopping the channel using an old token distribution because if a malicious user presents an old contract in the ledger to stop the channel, an honest user will use the secret previously revealed and get all the tokens. To work, honest users must periodically check the blockchain to detect malicious behaviors, just like in commit-chains. Also, the system assumes that there is a maximal delay under which we are sure that a transaction is validated by the blockchain network. Otherwise, an honest user cannot prove that a malicious user is closing the channel with an invalid state.

To speed up the channel closing, A and B can agree on a final contract signed by both users without conditions. Once this transaction is added to the ledger, the token distribution is immediately applied.

Hashed Timelock Contract (HTLC) are a special kind of smart-contracts that allows users to execute an atomic swap [39]. Atomic swaps are a kind of exchange where if all users conform to the protocol, then all exchanges take place, and if a user deviates from the protocol, then no conforming user ends up worse off, even against a coalition of malicious users.

[39]: Herlihy (2018), 'Atomic cross-chain swaps'

The secret sharing scheme used by HTLC can be extended to allow tokens exchange through several channels. For instance, when A wants to pay C, but A only has a channel with B, and C only has a channel with B. In this case, it is possible to create a virtual channel between A and B. Channels opened with a common transaction with two ends of the channel, as we presented earlier, are called ledger channels. The virtual channel is a sequence of ledger channels that connects two users without a common ledger channel [40, 41]. The ledger channels create a network of virtual channels that helps the transaction exchanges between users and increases the scalability of the blockchain network.

[40]: Dziembowski et al. (2018), 'General state channel networks'
[41]: Dziembowski et al. (2019), *Multi-party virtual state channels*

3.2.3 Pros/cons

Table 3.2: Pros and cons for state channels

Pros	Cons
- state channels reduce the number of transaction stored in the ledger.	- stopping the channel and retrieving the locked tokens can take time.
- transactions are instantly validated in the channel.	- it is possible that there is no virtual channel to interconnect two users.
- transactions have almost no fees.	- honest users have to periodically check the blockchain to detect malicious users closing the channel with an invalid state.

In this presentation of state channels, we focus on Lightning Network, but the concept used in this proposition has been generalized to exchange any kind of transactions [40]. When the channel is used to pay users, the channel is sometimes referred to as a payment channel, and in the general case, it is referred to as a state channel.

[40]: Dziembowski et al. (2018), 'General state channel networks'

We summarize the pros and cons of commit-chains in Table 3.2. Just like commit-chains, state channels increase the transaction throughput of the blockchain network. The transactions exchanges in the state channel are immediately applied in the state channel. Also, it reduces the number of transactions stored in the ledger. The ledger will only register the first state, when the channel is open, and the last state, when the channel is closed, and not a history of all the intermediary states.

There is also a financial interest in state channels. If channel users exchange frequently transactions between them, they will apply no fees on the transactions exchanged between them. In the case of virtual channels, the intermediaries used to interconnect two users can claim some fees, but the total number of fees cannot exceed the fees for a transaction in the ledger. Otherwise, the transaction will be performed directly in the ledger.

An issue with state channels is the possible delay required to stop the channel. If both users did not agree on a final transaction, when a user closes the channel by presenting the last exchanged contract, users have to wait for the timeout before retrieving their tokens on the ledger. This can be an issue for users with a small number of tokens who quickly need to use their tokens in the ledger.

Also, in the case of virtual channels, there is an issue with the availability of the channel. If A and B want to exchange tokens by going through several intermediary channels, because there is no direct channel between them, they have to check first if there are enough tokens available on all the intermediary. If even one intermediary does not have enough tokens, the transaction cannot take place. Thus forcing A and B to open a direct channel between them if they frequently exchange transactions or share the transaction directly in the distributed ledger.

3.2.4 Usage

In the case of Lightning Network, it truly increases the throughput of Bitcoin. For instance, in 2021, El Salvador declared Bitcoin as a legal tender in the country [42]. To help the citizens in using this new currency, the government used a state application called Chivo and created Lightning channels. Thus allowing the citizens to use Bitcoin in their everyday transactions. We will not develop the legal questions that this adoption raises, but we will underline the fact that state channels allowed the Bitcoin network to handle the everyday transactions of a whole country.

[42]: Alvarez et al. (2022), *Are Cryptocurrencies Currencies? Bitcoin as Legal Tender in El Salvador*

Allowing the blockchain network to handle a large throughput of transactions could be useful in Internet of Things (IoT) environments where devices generate a lot of new data. From this observation, we can ask ourselves if the blockchain can be used to improve security in IoT. We are going to observe this in the next chapter.

Blockchain and IoT 4

The Internet of Things (IoT) includes all the applications where physical objects share data on the Internet or in other communication networks. These objects catch data from their environments and share it with an application. For instance, they can be sensors monitoring the temperature of a specific environment. The application will then use these data to adapt itself to the environment.

These objects can be constrained devices and they can be part of constrained networks. By constrained, we mean that some characteristics that are otherwise pretty common are not present anymore. For instance, the reliability and the bandwidth of the network is low , the storage and computing resources are small, or we cannot use a complete TCP/IP stack on the device. Thus making it difficult to directly use well-known Internet protocols in IoT and force the creation of new protocols. For instance, IEEE 802.15.4 and 6LowPan [43] explain how to exchange IP messages in a Low Rate Wireless Personal Area Network (LR-WPAN), networks where members have a battery, low wireless range, and low bandwidth.

[43]: Montenegro et al. (2007), 'Transmission of IPv6 packets over IEEE 802.15. 4 networks'

Since the devices monitor the environment, data can be sensitive, hence must be transported and used securely. For instance, if a thief hacks the devices or the application monitoring a house, he can know when the house is empty and execute a burglary. To add security in those applications, several propositions are using Distributed Ledger Technologies (DLT). In this chapter, we review some of these propositions and find out when blockchain can breally help IoT applications.

4.1 Blockchain nodes on constrained devices

In Chapter 2, we explain that some of the current challenges in blockchains are the size of the complete ledger and the high use of computational power to execute the Proof of Work (PoW). With constrained devices, these challenges add even more difficulty. To underline this fact, we will use the RFC 7228 [44] that defines different classes of constraint devices based on their memory resources. These classes are summarized in Table 4.1.

[44]: Bormann et al. (2014), 'Terminology for constrained-node networks'

Table 4.1: Classes of constrained devices

	RAM	Flash
Class 0	<10.24 KB	<102.4 KB
Class 1	∼10.24 KB	∼102.4 KB
Class 2	∼51.2 KB	∼256 KB
No class	>51.2 KB	>256 KB

We can observe that the 3 classes of constrained devices have very few memory resources. To compare this classes with the resources required to be a distributed ledger node, we indicate in Table 4.2 the required configurations for 3 DLT clients. We selected Bitcoin with Bitcoin Core [45], Ethereum with Geth [46], and IOTA with Hornet [47].

[45]: Bitcoin Project (2022), *Bitcoin Core requirements*

[46]: Ethereum Foundation (2022), *Geth requirements*

[47]: IOTA Foundation (2022), *Hornet requirements*

Table 4.2: Requirements for distributed ledger clients

	Hard drive	RAM
Bitcoin Core	350GB	1GB
Geth (Ethereum) light node	400GB	4GB with SSD 8GB with HDD
Geth full node	6TB	′
Hornet (IOTA)	not specified	8GB

Bitcoin and Ethereum are the two most popular blockchain networks, Bitcoin because it is the first-ever blockchain, and Ethereum since it is the first blockchain implementing smart-contracts writable with a Turing-complete language. We choose to add these two blockchains because they appear in many propositions. On the other hand, IOTA is not a blockchain. It is a DLT that does not rely on a chain of blocks but relies on a Directed Acyclic Graph (DAG) of transactions. We will not discuss in detail how this DLT works, but we add it because the purpose of IOTA is to be used by IoT devices.

We can observe that there is a severe difference between the requirements of DLT clients and the constrained classes presented in RFC 7228. No constrained classes is having the requirements for at least one client. The least demanding client in RAM, Bitcoin Core, demands 10^5 the RAM of the least constrained class, Class 2. The difference is the same if we compare them with the flash memory required.

From this observation, adding IoT devices directly in the distributed ledger network became an unrealistic assumption. Still, there are several works doing so [48–52]. For instance, in [50], the blockchain is used as a common database between the device of the same network. In this case, the benefit from the blockchain is limited compared to the cost of being a blockchain node for constrained devices. At one point in time, the device will not have the capacity to store the ledger and it will be impossible to add new information to it.

[48]: Gupta et al. (2018), 'The applicability of blockchain in the Internet of Things'
[49]: Singh et al. (2018), 'Blockchain: A game changer for securing IoT data'
[50]: Pinno et al. (2018), 'ControlChain: Blockchain as a Central Enabler for Access Control Authorizations in the IoT'
[51]: Devi et al. (2019), 'Integration of Blockchain and IoT in Satellite Monitoring Process'
[52]: Sharma et al. (2017), 'Block-VN: A distributed blockchain based vehicular network architecture in smart city'

Based on the storage resource required, being a node in a distributed ledger network and being a constrained device seems unrealistic with the current requirements for the distributed ledger. If we want to use blockchain, or in general DLT, to improve security in IoT applications, we need to put the node elsewhere in the topology.

4.2 Blockchain nodes on unconstrained devices

It is unrealistic to add constrained devices directly to the blockchain network because they do not have the minimal requirements, so we need to define where it seems possible to add blockchain nodes. In Figure 4.1, we represent the two positions that seems realistic.

The first solution is to add the gateway of the IoT network in the blockchain network. The gateway is the connector between the IoT network and the Internet. All messages coming in or out of the IoT network go through it. Because of this responsibility, the gateway is usually more powerful than the member of the IoT network. It can be an unconstrained device that could match the requirement for joining the blockchain network. If

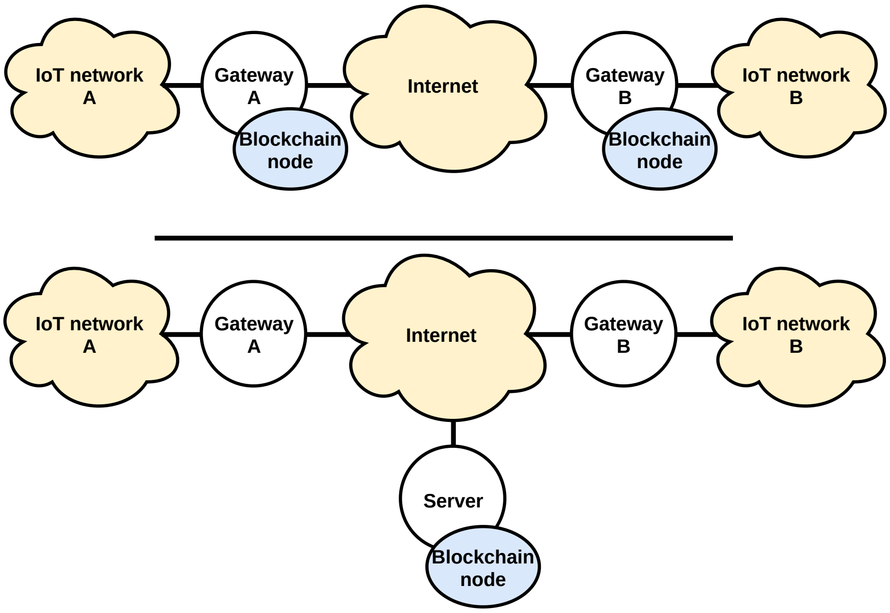


Figure 4.1: Blockchan node positioning in IoT topology

the gateway is a blockchain node, the constrained devices can directly contact the blockchain by reaching the gateway.

The second solution is to use a server as a blockchain node. Having a server with the requirements for the blockchain network is a realistic assumption. In this case, if the constrained device wants to interact with the blockchain network, the gateway will act as an intermediary. The gateway will contact the blockchain node on the behalf of the constrained device and forward the result to the constrained device if necessary.

These two node positionings are present in the literature and we will see how they are used.

4.3 Blockchain usage

There is several propositions using DLTs in IoTs, and they respond to different problems. In this section, we will group the propositions based on how they use DTLs and on the problem they want to solve.

4.3.1 Database

In several propositions, the blockchain is used like a common storage space for IoT data [53–58]. In [54, 55], data generated in LoraWAN networks are stored in a blockchain. Each data is sent to the blockchain network by the gateway. By storing data in the ledger, these propositions ensure that data cannot be modified and can be trusted. From the same idea, in [56], the authors present a solution where data are stored in a cloud storage system and the data fingerprints are stored in a smart-contract to prove data authenticity. In [58], devices send data to a smart-contract and, based on this data, the contract tells if some devices have to execute a specific command. In this case, putting a light on or off.

[53]: Huang et al. (2019), ‘B-IoT: Blockchain driven internet of things with credit-based consensus mechanism’
[54]: Lin et al. (2017), ‘Using blockchain to build trusted LoRaWAN sharing server’
[55]: Niya et al. (2019), ‘Adaptation of Proof-of-Stake-based Blockchains for IoT Data Streams’
[56]: Liu et al. (2017), ‘Blockchain Based Data Integrity Service Framework for IoT Data’
[57]: Dorri et al. (2017), ‘Blockchain for IoT security and privacy: The case study of a smart home’
[58]: Huh et al. (2017), ‘Managing IoT devices using blockchain platform’

In all these solutions, each new data generated in the IoT network creates at least one transaction in the ledger. This usage of the blockchain raises several questions on the utility of blockchain in IoT. First, in large-scale adaptation, this kind of proposition will face scalability problems with storage resources and network's throughputs presented in Chapter 2. Adapting these propositions with off-chain protocols like the one in Chapter 3 can be a solution. Second, are the transactions useful in the long term? As we said, in DLT, data cannot be updated. The nodes will store the whole ledger as long as they are part of the network. So, data lifetime in the ledger has to be compared with IoT data utility in the application. Most of the time, IoT data are useful in the moment for the application, and maybe sometime later to have an overview of the application execution, but are they useful after that? For instance, is it important to keep the temperature catch by a sensor 10 years after the measurement? If the answer is no, then storing all IoT data, or data fingerprint, in the ledger seems not like a good idea.

Also, this usage of the blockchain can create privacy problems. For instance, in [58], data sent by the devices are used by the contract to give orders to other devices. This decision is visible to all blockchain users. For instance, if the decision is to cut the lights off and shut down radiators in a house, we can assume that the house is empty. Which is useful information for burglars. When designing a protocol relying on a blockchain, it is important to assume constant eavesdropping to resolve this privacy issue because transactions are public, so malicious users can read them. Sensitive data have to be encrypted. Otherwise, they can be used by malicious users.

4.3.2 Resource access management

DLT are used in many propositions for resource access management in IoT [59–63]. The resource can be writing or reading accesses to an IoT device or network. For instance, to retrieve some measurements or to execute a command on the device. In fact, the user can also be an IoT device. In this case, the resource can be a cloud storage system on which the device wants to store data.

[59]: Novo (2019), 'Scalable access management in IoT using blockchain: A performance evaluation'
[60]: Novo (2018), 'Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT'
[61]: Zhang et al. (2019), 'Smart contract-based access control for the internet of things'
[62]: (2018), 'IoTChain: A blockchain security architecture for the Internet of Things'
[63]: Steichen et al. (2018), 'Blockchain-Based, Decentralized Access Control for IPFS'

Without using a DLT, this kind of problem is resolved with an authorization authority. The user asks the authority for a token, and the authority shares a token with the user if the access is granted. The user will then present the token to the resource gateway and, after checking the validity of the token, will have access to the resource. The issue with this model is the authorization authority. What if this authorization authority is malicious and does not respect the access rules for the resource? By giving tokens to an unauthorized user, for instance. The authorization authority lacks transparency, and that is why some propositions replace it with a blockchain.

These propositions share a common process on how to access the resource. This process is represented in Figure 4.1. Instead of using a server as an authorization authority, a smart-contract is used to generate access tokens. The contract is written by the resource owner. It describes the access rules of the resource. Once a user requests a token, the contract checks if the request matches the rules. If it is the case, the token is

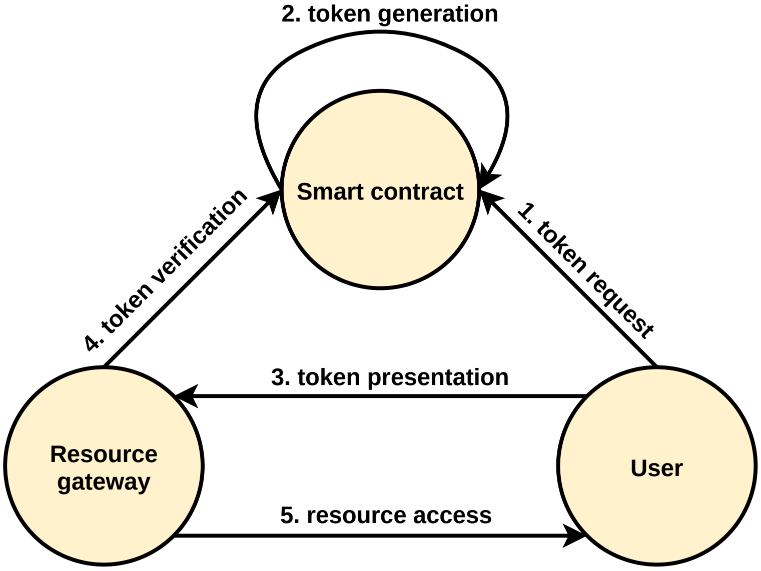


Figure 4.2: Resource access management with smart-contracts

created in the smart-contract. The user presents the token to the resource gateway, and once the verification is complete, the resource access is granted to the user.

To verify the token's validity, the gateway has to execute a double verification. First, it has to verify that the token was created by the smart-contract, and so is present in the blockchain. Second, since tokens are visible to all users with access to the blockchain, the resource gateway has to verify the identity of the entity presenting the tokens. To do so, it checks if the public key used for the token request is the same as the public key of the user presenting the token.

Removing the central authorization has several benefits. One is the availability of the authorization authority. With a central server, you can handle crashes with replication and high-availability solutions, but what if the server leaves the system and goes offline? In this case, access to the resource is blocked because it is impossible to generate new tokens. On the other hand, the smart-contract is added to a blockchain. As long as the user and the resource gateway can reach one node of the blockchain network, tokens can be requested and verified. The token generation can only be stopped if there is no node executing the consensus algorithm, but this assumption seems unrealistic because nodes have an incentive for being part of the consensus.

The second benefit is the transparency of the token generation process. Since the access rules are written in the smart-contract, it is impossible to bypass them when requesting a token, because invalid transactions will not generate a token and the blockchain nodes verify the transaction validity. For this reason, it is impossible to have a malicious authorization authority when using a smart-contract, under the assumption that the blockchain network is trustable. Also, in some propositions, it is possible to update the access rules [59, 61]. For these propositions, the blockchain also ensures that only the correct users can update the access rules. The code of a smart-contract cannot be changed, but the variables inside the contract can. If the rules are stored in variables, then they can be updated with contract functions.

The transparency of the token generation process also simplifies conflict resolution, because it adds audibility to the token creation. If the resource owner detects incorrect resource access, it knows that the gateway behaves incorrectly, because an incorrect request will not have generated tokens, if we assume that the access rules in the contract are correct. Also, if a user presenting a token for a given resource can prove that it receives another resource from the gateway, or the gateway denied the access, then the presence of the token in the blockchain is proof that the gateway should have granted the access to the correct resource, proving an incorrect behavior from the gateway.

Notice that if the resource gateway, the authorization authority, and the users have the same administrator having access to the private key of each user, this usage of the blockchain lacks utility, because it will add trust between entities managed by the same person.

4.3.3 Reputation system

Blockchain users generate transactions, and these transactions are eventually added to a block by the nodes. Like we said earlier, all these transactions are public, they can be read by every blockchain user, and cannot be modified. In some situations, by analyzing these transactions, we can have an overview of the user's behaviors in the application and we can create a reputation system based on auditable behaviors, and not on other users' feedback.

Having a reputation system means that there are possible conflicts between users. This means that some users can disagree on a specific event. For instance, when the sender shares data fingerprints in the ledger, like in [56], if the receiver can prove in the ledger that the received data have a different fingerprint, then it proves that the sender made an error. From the same idea, in access resource management, if the user can prove that it did not receive access to the resource with a correct token, or receive access to another resource, it can prove that the gateway behaves incorrectly. So resource owners will not put resources behind the incorrect gateway. In other words, having a good reputation means that new users will most likely interact with you.

In [64], the authors present a witness model for service level agreements. To ensure that the service has the correct quality, the service provider asks for a sorting smart-contract to create a witness committee. The committee is created by randomly selecting users willing to be part of a committee.

[64]: Zhou et al. (2019), 'A Blockchain based Witness Model for Trustworthy Cloud Service Level Agreement Enforcement'

The service provider presents the committee, the service requirements, and payment to another smart-contract. Once the contract has checked the committee and once the service user agrees on the requirements, the service is available. During the service's execution, the committee verifies if the requirements are respected. If a witness detects an issue, it reports it to the smart-contract. If the number of error reports reaches a certain threshold, the smart-contract considers that the service provider did not respect the requirements, and the service user receives compensation from the service provider's initial payment. No matter the result of the service, the witnesses receive also a payment.

Based on their behaviors, witnesses have a reputation in the sorting smart-contract. For instance, lazy witnesses who never declare a violation of the service requirement will lose some reputation if the service is considered violated. The process also applies to witnesses who always declare a service violation, even when the service is considered correct. When this reputation reaches 0, the lowest score possible, the witness cannot be selected anymore by the smart-contract. The issue is that this malicious witness can register itself back in the sorting smart-contract with a new key, and a new reputation score.

In DLT, users are identified with a public key, so malicious users can generate an infinite amount of keys to create new identities and reset the reputation score. To be effective, reputation systems using DLT must counter Sybil attacks by forcing the user to invest resources, just like the PoW. For instance, in the case of the witness committee, witnesses could lock some money in the sorting smart-contract to be eligible for a committee. If a user's reputation reaches 0, the locked money will not be retrievable by the user. This money could pay other members of the committee, forcing them to cooperate to detect malicious witnesses. Here, the money is a counter to Sybil attacks, because it is a finite resource. Each user owns a certain quantity of money, but no user has an unlimited amount.

4.3.4 Conclusion

DLT can be used to add security in IoT applications, but the application must take into consideration several points. First, since being a node is too demanding for IoT devices, it is impossible to add constrained devices as full nodes. The nodes must be placed elsewhere in the system, and constrained devices can go through a trusted intermediary to contact the ledger if they need to. Second, all data in the ledger are publicly auditable and unmodifiable. It can be interesting to track users' behavior and create a reputation system, but sensitive information must not be shared this way because malicious users could use them. In the rest of the thesis, we will focus on one specific part of IoT: publish/subscribe communications.

Publish/subscribe 5

5.1 General concept

The publish/subscribe paradigm is a communication model used to exchange information. In this model, there are 3 kind of entities: publishers sharing events; subscribers receiving events; the broker is an intermediary, a central entity in a star topology. In Figure 5.1, we represent a publish/subscribe topology.

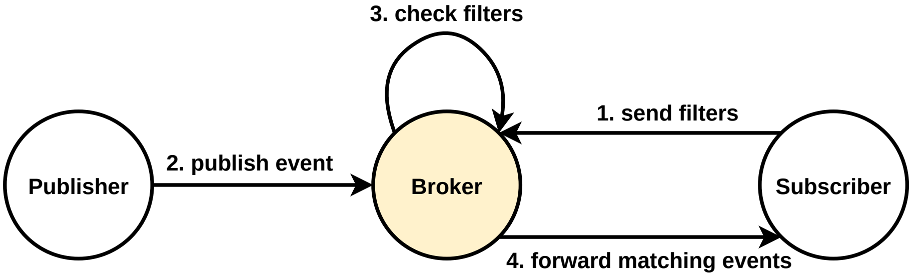


Figure 5.1: Publish/subscribe architecture

To notify the broker of its interest in specific events, a subscriber sends filters to the broker. The set {subscriber,filters} is called a subscription. When the broker receives events from a publisher, it checks all the subscriptions to see if events match the filters. The broker will then forward events to all matching subscribers. Events can represent any kind of data.

The interest of this model for IoT scenarios, or large-scale one-to-many communications, is the loose coupling between the publishers and the subscribers [2]. This loose coupling takes place on 3 levels:

- Space decoupling: the publisher and the subscriber do not need to know each other.
- Time decoupling: the publisher and the subscriber do not need to be online at the same time.
- Synchronization decoupling: the publisher is not blocked by the subscriber, it can always produce events. The subscriber is asynchronously notified by the broker.

This loose coupling property is interesting in IoT environments for the constrained devices. For instance, the time decoupling property is useful to spare constrained device batteries. The constrained device periodically calls the broker to check if there is any new event, or share new events, without knowing the states of other devices. The space decoupling and synchronization decoupling properties increase the scalability of the network: users can join or leave by just notifying the broker and not every publisher and subscriber with whom it interacts.

The publish/subscribe paradigm can be used to connect constrained devices with IoT services or other devices [65, 66]. This communication

[2]: Eugster et al. (2003), 'The Many Faces of Publish/Subscribe'

[65]: Sun et al. (2013), 'A low-delay, lightweight publish/subscribe architecture for delay-sensitive IOT services'
[66]: Amoretti et al. (2020), 'A scalable and secure publish/subscribe-based framework for industrial IoT'

model can also be adapted for static or moving devices. For instance, in [67], data from mobile devices are collected by a broker host on a cloud system.

[67]: Antonić et al. (2016), 'A mobile crowd sensing ecosystem enabled by CUPUS: Cloud-based publish/subscribe middleware for the Internet of Things'

As we said, the subscriber presents filters to the broker. There are two kinds of publish/subscribe protocol, depending on the kind of filter they use: topic-based protocols and content-based protocols. In topic-based protocols, each event from the publisher is linked with a topic, a logical name. In this case, the filters given by the subscriber are the names of the interesting topics. For instance, "building/6th-floor/temp" can represent the temperature catched by sensors in the 6th floor of a building. On the other hand, in content-based protocols, the filters are on the content properties. For instance, values in a specific interval.

The most famous topic-based publish/subscribe protocol is MQTT [68] (Message Queuing Telemetry Transport). The purpose of this protocol is to be easily implementable on the client's side with the minimum resource consumption possible when used. To do so, the protocol has a short message format and the computation complexity is located on the broker.

[68]: Andrew Banks et al. (2019), *MQTT Version 5.0*

In the rest of this chapter, we will observe how security is added to this communication paradigm with and without using DLT.

5.2 Without DLT

5.2.1 End-to-end encryption

End-to-end security is a method used in several propositions [69–72] 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 by malicious entities eavesdropping on the links between the broker and the publishers/subscribers. To achieve this, these propositions rely on encryption mechanisms. By encrypting data, security is achieved between the publisher and the subscribers because the subscribers are the only entities in the system capable of reading published data. In some solutions, the payload can also be signed to prove message integrity, like in [72].

[69]: Dahlmanns et al. (2021), *Transparent End-to-End Security for Publish/Subscribe Communication in Cyber-Physical Systems*
[70]: Kumar et al. (2019), 'Jedi: Many-to-many end-to-end encryption and key delegation for IoT'
[71]: Borcea et al. (2017), 'PICADOR: End-to-end encrypted Publish–Subscribe information distribution with proxy re-encryption'
[72]: Pallickara et al. (2006), 'A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems'

Sometimes, these propositions broke the loose coupling property of publish/subscribe communication. In [71], 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. Since the derived key uses the publisher's private key, in a real application, the publisher will be the only entity capable of creating this key. It seems unrealistic that a third party will create the derived key because the private key is a secret that the publisher does not want to share. To generate the key, the publisher needs the public key of the subscriber, so it needs to know the identity of the subscriber. This breaks the space decoupling property stating that the publisher and the subscriber do not need to know each other. To prevent this issue, a symmetric key can be created for each topic, like in [69, 72]. The publisher and the subscriber do not need to know each other identity, but they only need the key to exchange data.

The critical part of end-to-end security methods is the key sharing process. From an initial phase where the publisher and the subscriber have no common key, we need to create a common key between the two users. To execute this operation, these propositions introduce a key managing authority, but these processes still have an issue: they are broker-dependent.

In publish/subscribe environment, all exchanged messages go through the broker, because it is in the middle of a star-topology, as represented in Figure 5.1. This means that the publisher and the subscriber must trust the broker to which they are connected. If the broker cannot be trusted, it means that the broker will drop or indefinitely delay messages from or for the connected user. For end-to-end encryption, it means that in this case, the key sharing process could not finish. We cannot guarantee that the key can be shared.

If the publisher or the subscriber does not trust the broker, they have to connect themselves to another broker. In the example presented above, the publishers and the subscribers are connected to the same broker. This broker can be load balanced and high available by being executed on a set of servers but, on a application level, it is only one broker. In fact, it is possible to have an publish/subscribe architecture where several brokers have to cooperate to share data. If the publisher and the subscriber are connected to two different brokers, we have a peer-to-peer, or a distributed system, where we need to introduce security between the brokers. This is what we are going to see in the next section.

5.2.2 Distributed publish/subscribe

In this section, we discuss distributed publish/subscribe propositions, propositions where several brokers have to cooperate to share data between publishers and subscribers. There is an extensive part of the community answering how the interconnections between the brokers should look like and how to route events through the network of brokers in the most optimal way [73–77]. Each proposition answers these questions in different kinds of networks. We consider these works out of our scope and we focus ourselves on propositions looking at adding security in distributed publish/subscribe because we think that DLT can add something in those propositions.

There are several works answering security issues in distributed publish/-subscribe [78–80]. Those works resolve authorization issues, message privacy, integrity concerns, and other security problems. To do so, they exchange encryption and signature keys between the publishers, the subscribers, and the brokers.

For instance, Srovatsa et al. [79] presents a solution called Eventguard. It explains how to secure the publish/subscribe system from unwanted actions, like flooding the network with incorrect subscriptions or publications, by using a trusted meta-service. This meta-service generates keys linked with each topic and certificates used by users to identify themselves in the system. To handle dropped messages from potential malicious nodes and keep connectivity, the protocol assumes that a significant fraction of nodes are nonmalicious and uses multi-path sending. The solution is interesting but is, on a logically level, centralized. The

[73]: Cao et al. (2004), 'Efficient event routing in content-based publish-subscribe service networks'

[74]: Bianchi et al. (2010), 'Stabilizing distributed R-trees for peer-to-peer content routing'

[75]: Shah et al. (2004), 'Efficient dissemination of personalized information using content-based multicast'

[76]: Voulgaris et al. (2005), 'S UB -2-S UB : Self-Organizing Content-Based Publish and Subscribe for Dynamic and Large Scale'

[77]: De Araujo et al. (2017), 'A Publish/-Subscribe System Using Causal Broadcast over Dynamically Built Spanning Trees'

[78]: Pallickara et al. (2003), 'A security framework for distributed brokering systems'

[79]: Srivatsa et al. (2011), 'Eventguard: A system architecture for securing publish-subscribe networks'

[80]: Dini et al. (2009), 'On securing publish-subscribe systems with security groups'

meta-service creating all the cryptography keys used in the system must be trusted, hence its name.

Pallickara et al. [78] propose a security framework to ensure authentication and maintenance of identity, scalable topic security, and message security in a peer-to-peer publish/subscribe system with topics. To handle the security, a central authority called the Key Management Center (KMC) shares the topic keys. For instance, if a publisher wants to share data on a certain topic, it requests the topic key to the KMC. If the publisher is authorized to do so, the topic key is encrypted with the public key of the publisher and sent to it. If a malicious user tries to share data on a topic without authorization, it will be eventually detected by an honest broker. Each broker checks if the publisher is authorized to share data when they route messages in the peer-to-peer network.

These solutions rely on a central authority to generate the security mechanism used in the network of brokers. The integrity of this entity is key. For this reason, DLT can be interesting to support this central authority, since it is transparent and tamper-proof. It can add trust in this authority, just like in the resource access management use case presented in Chapter 4.

5.3 With DLT

Blockchains, or DLT in generals, are used in different publish/subscribe propositions [3, 4, 81].

Trinity [3, 4] is to our knowledge, the first publish/subscribe protocol using blockchain. The architecture of the proposition is represented in Figure 5.2. In this solution, brokers are nodes in a blockchain network. They use the blockchain to replicate data and store them in an immutable way. When a client publishes data by sending it to its broker, the broker forwards it in the blockchain network. The data eventually appears in a block so that other brokers retrieve this information by reading the blockchain and forward the published data to their local subscribers.

[3]: Ramachandran et al. (2019), 'Trinity: A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence'
[4]: Ramachandran et al. (2018), 'Trinity: A Distributed Publish/Subscribe Broker with Blockchain-based Immutability'
[81]: Lv et al. (2019), 'An IOT-oriented privacy-preserving publish/subscribe model over blockchains'

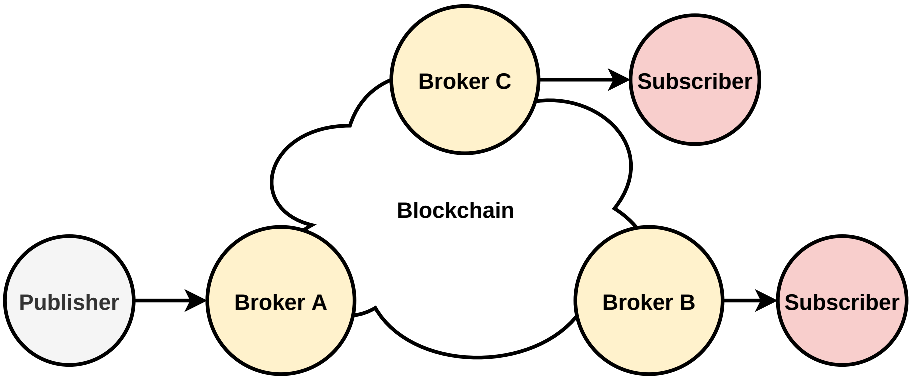


Figure 5.2: Trinity

This interaction between a publish/subscribe protocol and a blockchain creates a secured system where we can trust the brokers because each received data can be linked to the entity that published it. However, this approach does not scale for the same reasons as some IoT propositions presented in Chapter 4.

First, writing on the blockchain is not free. Each new transaction has a cost. With this price, the network can stay alive by paying blockchain nodes for their work. Over time, if a lot of data are published, this kind of solution could become expensive for the brokers or the clients.

Second, this approach does not scale well with the number of users. Indeed, since blocks have a maximum number of transactions and are generated at a constant rate with the consensus algorithm, blockchain networks can process a limited amount of transactions per second. This value depends on the implementation and cannot be changed. For instance, the Bitcoin network processes 7 transactions per second, which is far from enough for sharing IoT data, where you have millions of sensors. On top of this, since transaction validation takes time, it adds delay in data delivery. The other brokers have to wait for the data to be added to the ledger before forwarding it to the local subscribers.

After Trinity, other variants were presented [81], but keeping this extensive use of blockchain for each newly published data. Trinity proves that it is possible to implement a publish/subscribe protocol using the blockchain, but it shows the limitation of overusing the blockchain. To create a scalable protocol, we need to reduce as much as possible the number of transactions added to the ledger during the communication and, to do so, we can inspire ourselves from a data payment protocol.

Bo et all. [82] also present a publish/subscribe application using blockchain named HyperPubSub. They select an online photo trading system as a use case for the proposition. The photos are stored in a data base and the meta data are stored in the ledger. The publishers are the photographers and the subscribers are customers. The blockchain acts as a broker between them where the trade between the customers and the photographers happen. This proposition uses a private blockchain and functions specific at Hyperledger Fabric [83], a blockchain implementation. It reduces the scope of applications of the proposition.

[82]: Bu et al. (2019), 'HyperPubSub: Blockchain based publish/subscribe'

[83]: Androulaki et al. (2018), 'Hyperledger fabric: a distributed operating system for permissioned blockchains'

5.4 Conclusion

The publish/subscribe paradigm can benefit from the blockchain. The current solutions using the blockchain prove the utility of the technology but they suffer from the same limitation as other IoT solutions using the blockchain. We focused on publish/subscribe solution using the blockchain to share data but in some cases, like in HyperPubSub, the blockchain is also used to sell data. In the next chapter, we will have a closer look at those propositions.

Data payment protocol 6

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 has to execute an atomic swap [39] between the data and the payment. As presented in Chapter 3, 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 is malicious and tries to deviate from the protocol, then all honest users cannot end up in a situation worse than the one before executing the swap.

Blockchain and cryptocurrencies can be used for creating a secure data payment 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 IoT 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. In this chapter, we take a closer look at some data payment protocol propositions using blockchain.

6.1 SDPP

[84]: Radhakrishnan et al. (2018), 'Streaming Data Payment Protocol (SDPP) for the Internet of Things'

Streaming Data Payment Protocol (SDPP) [84] is a data exchange protocol that allows a data provider to receive monetary payments from consumers. To set up the data stream, the consumer and the provider exchange an order that contains the topic selected by the consumer and the quantity of data awaited. This order is stored in a blockchain, but the data stream then occurs through a TCP session. In this order, the provider and the consumer agree on two important parameters: the total amount of data D and the granularity of payment K.

Once the order is added to the blockchain, the data stream can begin, and the provider will share data in windows of size K. We represent the process for one window in Figure 6.1. Each data is sent through the TCP session by the provider and is acknowledged by the consumer. During the exchange, between each window of size K, the provider sends an invoice, stored on the blockchain, to the consumer. Before sending the next K data, the provider waits for the payment in crypto-currencies by the consumer. After the provider sent D data by windows of size K and the consumer paid the final window, the session is closed. The consumer can stop earlier the session if needed by notifying the provider. In this case, the consumer will only pay for the data already received.

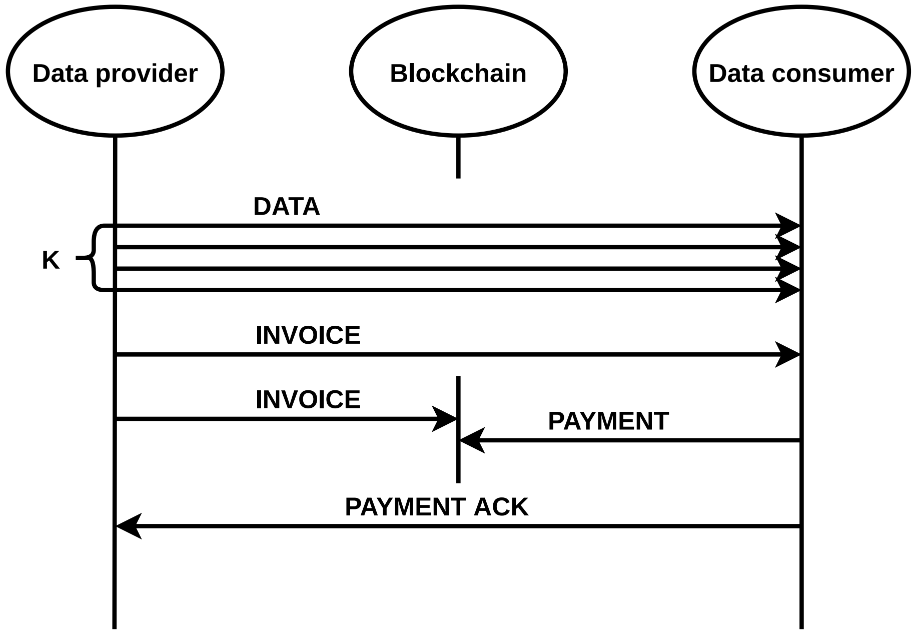


Figure 6.1: Sending of a data window in SDPP

SDPP has two interesting aspects. The first one is auditability. The orders, the invoices, and the payments are stored on the blockchain, which allows any third party to check the evolution of the sessions. Also, the protocol reduces the number of messages sent to the blockchain, since the data stream is exchanged in a TCP session. Also, the protocol is not specific to one blockchain, it can run on many blockchains since it does not require smart-contracts. The crypto-currencies used to pay can be on the same blockchain as the invoices or the users can use two different ledgers.

On the other hand, the protocol has two major downsides. First, nothing prevents the data consumer to leave the system without paying. At the end of the window, the provider sends an invoice to the consumer, and a copy of this invoice is also sent to the ledger. The consumer can go offline at this moment and never pay the K data. To reduce the loss from an unpaid window, the provider can reduce the size of the window but it will increase the impact of the second downside of the proposition. Indeed, between each window, the consumer has to pay in cryptocurrencies to the data provider. Cryptocurrencies are blockchain applications and they inherit from this blockchain property: a transaction takes time to be added to the ledger. The time needed depends on the blockchain. For instance, in Bitcoin, since the chain can fork, once the transaction is in a block you have to wait around one hour before being sure that the transaction is in the main branch of the chain. Applying this property on SDPP means that between windows, the system is on hold for one hour because the provider waits for the confirmed payment before sending the next window. This issue can prevent the usage of SDPP for real-time data.

D. Chen et al. [85] 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. [86] proposed an implementation of SDPP on a Raspberry Pi, but it keeps the payment issue of the protocol with

[85]: Chen et al. (2019), 'PayFlow: Micropayments for bandwidth reservations in software defined networks'

[86]: Nakada et al. (2021), 'Implementation of Micropayment System Using IoT Devices'

the data window.

6.2 Publish/subscribe propositions

Ramachandran et all. [87] authors present a system of payment for publish/subscribe protocol called PPSP (Publish-Pay-Subscribe Protocol). In this protocol, the publisher delegates the payment handling functionality to the broker.

[87]: Ramachandran et al. (2019), 'Publish-pay-subscribe protocol for payment-driven edge computing'

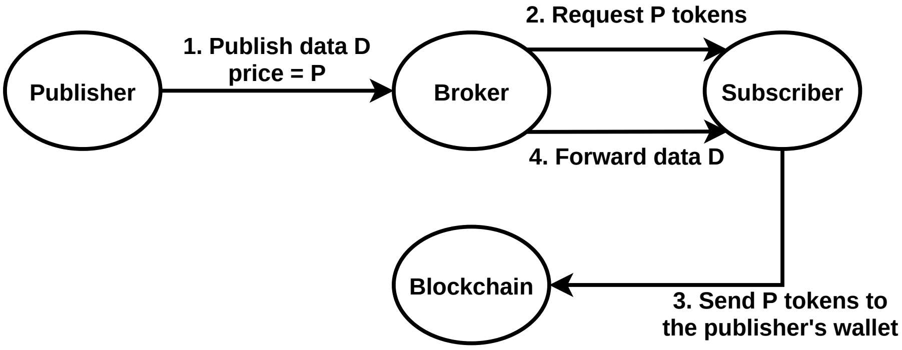


Figure 6.2: Data publication with PPSP

In Figure 6.2, we represent the data publication process with PPSP. The protocol used topics to filter data from the publisher. To open a subscription, the subscriber sends a topic to the broker. Once this process is complete, data can be sold.

Compared to the presentation of the publish/subscribe paradigm in Chapter 5, in this proposition, when the publisher sends data D to the broker, it will also share a price P. Then the broker will request this price from the subscribers. To pay, the subscriber sends the tokens to the publisher's wallet. When the payment is confirmed, the broker forwards the data to the subscriber.

Just like SDPP, the issue of this proposition is that the payment verification adds a delay before delivering each data. Instead of having a delay for each window, like in SDPP, in this protocol, each data is delayed. This delay can prevent the usage of this proposition for real-time data, where the subscriber needs to receive the data as fast as possible.

On the other hand, the subscriber cannot get data for free. The broker only forwards data when the payment is confirmed. Instead, the subscriber could pay for nothing. There is no guarantee that the subscriber will receive its data. Furthermore, this proposition is not blockchain specific. It does not use functions specific to one blockchain. For instance, Bu et all. [82] also present a payment protocol with publish/subscribe communication for digital assets but the proposition uses functions specific to Hyperledger Fabric [83].

[82]: Bu et al. (2019), 'HyperPubSub: Blockchain based publish/subscribe'

[83]: Androulaki et al. (2018), 'Hyperledger fabric: a distributed operating system for permissioned blockchains'

PPSP highlights the trust needed by the subscriber and the publisher to the broker. Nothing prevents the broker from requesting an incorrect price from the subscriber or even sharing data for free. This means that we have to assume that the broker is not malicious.

6.3 Conclusion

Crypto-currencies can give a precise financial value to small digital assets. It can be used to design data payment protocol and can be implemented with a publish/subscribe protocol. The issue is that the current solutions lack security or are specific to one blockchain implementation. There is no data payment protocol for publish/subscribe exchange that presents the security guarantees for the vendor and the buyer, reduces the delay implied by the blockchain, and is not specific to one DLT implementation. One of the propositions of this thesis is to propose one such protocol. We will present it in the next chapters.

SUPRA

General concepts 7

In the next chapters, we will present the first version of the main proposition of the thesis: Secured Update Protocol with Righterous Accusations. Before presenting in detail the protocol, we will explain the architecture in which we describe our protocol and a communication pattern used by the protocol.

7.1 Architecture

In this section, we explain the architecture in which our protocol will run. Figure 7.1 shows an example of architecture with 3 brokers, 2 publishers, and 2 subscribers. We will explain in more detail the figure in the next sections.

7.1.1 Distributed publish/subscribe

We place ourselves in a distributed publish/subscribe architecture. This means that there are several brokers, subscribers and publishers. Each publisher and subscriber connects itself to only one broker. We assume that publishers and subscribers trust the broker to which they are connected. It implies that, by assumption, if the broker receives an event for its subscriber, it will forward this event to it, and if it receives an event from its publisher it will forward this event to the local subscribers and to the other brokers interested in this event. If the publisher or the subscriber does not trust the broker anymore, it can connect itself to another broker.

There is no specific requirement for the publisher and the subscriber, aside from being capable of contacting its broker. They can be constrained or unconstrained devices. We will specify the requirements for the brokers later.

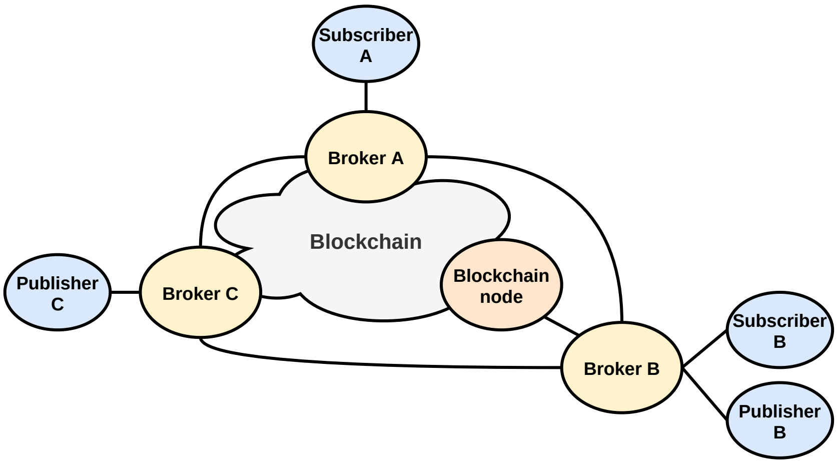


Figure 7.1: Architecture used by SUPRA with 3 brokers

7.1.2 Blockchain network

We assume that the brokers are reliably connected to the same blockchain network. This means that they are either nodes of this blockchain network or are reliably connected to a node or several nodes of this blockchain network. For instance, in Figure 7.1, Broker A and C are blockchain nodes and Broker B is connected to a blockchain node.

Like we said in Chapter 4, constrained devices cannot be blockchain nodes, with the current requirements for blockchain clients. This implies that the brokers of our architecture either have the requirements for being a blockchain node or have a reliable connection to a trusted blockchain node.

We make several assumptions about the blockchain used by the nodes:

- it takes at most a known delay $\Delta_{on-chain}$ to add a transaction in the ledger.
- transactions can be large enough to contain any messages of SUPRA.
- smart-contracts can be stored on the blockchain.

The assumption on the known delay $\Delta_{on-chain}$ also implies that the link between the broker and the blockchain node is synchronous. $\Delta_{on-chain}$ can be defined as the maximal time needed by the blockchain network to add a transaction to the ledger or as the sum of this delay and the maximal time needed to send a message from the broker to the blockchain node. It may seem like a strong assumption, but it is a known assumption used in off-chain solutions [36, 38] and multi-blockchain swap [39].

[36]: Poon et al. (2017), 'Plasma: Scalable Autonomous Smart Contracts'
[38]: Poon et al. (2015), 'The bitcoin lightning network'
[39]: Herlihy (2018), 'Atomic cross-chain swaps'

On the other hand, there are no assumptions about the consensus protocol used by the blockchain network and about the authorizations required to add new nodes to the network. This means that if the blockchain used a permissionless protocol like the PoW, new nodes and so new brokers can easily join the system. On the other hand, if the network is permissioned, adding new brokers can be more difficult because you need authorization from a superior authority.

If we put aside the assumption of the transaction size, blockchains like Ethereum or Hyperledger Fabric can be used in our architecture. On the other hand, Bitcoin cannot be used, since the language used to code smart-contract is not Turing-complete. It will be difficult to implement the smart-contract on this platform.

7.1.3 Trust between the brokers

We assume that the brokers do not trust each other. If brokers cannot trust each other, then using publish/subscribe protocol is impossible between them for applications where data delivery is important. Indeed, because both brokers cannot know if the other broker is purposefully ignoring the messages or if the link between the brokers is too noisy.

If we have a communication protocol that ensures data delivery and generates proof of these events, then we can create communication between untrusted brokers, since it will be impossible for malicious users to deny the messages. In case of conflict, the proofs of delivery can be presented to an third-party to detect which broker is wrong.

The blockchain can be an interesting tool to create such a type of communication protocol. Data immutability combined with public auditability can prevent malicious users from denying an event. For instance, by sending directly the message to the blockchain, a malicious user cannot deny the existence of the message. Also, the third-party can be implemented in a smart contract and have a code auditable for the users.

7.2 The use of Unidirectional Channels in Distributed Publish/Subscribe Architectures

In a distributed publish/subscribe architecture, without a central server/broker, each broker sends data from their local publishers directly to the brokers to which the subscriber are connected. The communication between one broker to another for each topic is unidirectional and is totally independent of the other communications between the other broker. Indeed, if there is a problem between a broker A and a broker B that breaks up their relationship, the communication between A and another broker should not be impacted. Since we are interested in having secure, reliable, and auditable communications, we have to make sure that each unidirectional channel has those properties, independently from the other channels.

This observation implies that the design of a distributed publish/subscribe architecture, and so of SUPRA, can be decomposed into two main parts. The first part is to define a unidirectional channel protocol that allows one broker to send messages to another broker, with strong guarantees. This protocol is defined independently from our global architecture but requires several assumptions. The second part is to define a global protocol, that allows a set of brokers to set up unidirectional channels on the fly, allowing brokers with subscribers to request data from brokers with publishers. Setting up a channel requires, among other things, verifying the condition of the unidirectional channel.

SUPRA is the combination of those two parts. In the rest of this chapter, we define the first part of our architecture, which is a unidirectional channel protocol. Then, in the next chapters, we define SUPRA, which dictates how brokers should interact to set up secure subscriptions and use our unidirectional channel for all data communications.

7.3 Unidirectional Channel with On-Off Chain Proof of Delivery

In this section, we present a new communication protocol, used by SUPRA, but can be of independent interest. The purpose of this protocol is to allow one broker to send messages to another one, with delivery guarantees, and such that each broker obtains proof for each of those guarantees. The communication protocol is a unidirectional channel, but the goal is similar to the bidirectional channels defined in the Lightning network [38].

[38]: Poon et al. (2015), 'The bitcoin lightning network'

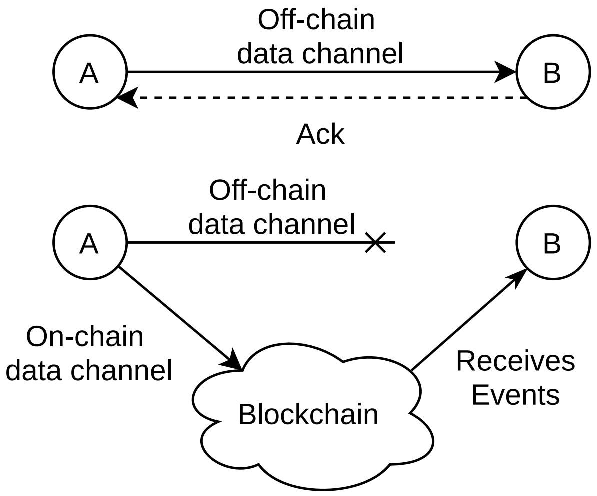


Figure 7.2: The two modes of communication of our unidirectional on/off-chain channel protocol

Informally, the broker with a publisher sends its messages to the broker with a subscriber using two methods, as illustrated by Figure 7.2. With the first method, the messages are sent like any other messages on the internet, directly to the broker with a subscriber, using its IP address, such messages are said to be sent *off-chain* because the blockchain is not used by this method. With the second method, the messages are sent on the blockchain, included in a block, and the broker with a subscriber indirectly receives the message by reading the blockchain. In this case, the messages are said to be sent *on-chain*.

Choosing which method to use depends on several parameters. By default, messages are sent off-chain, and if a broker tries to deviate from the protocol, or if there is a connectivity issue, messages will go on-chain. We now present what are the initial assumptions for our protocol to be used, how it works in more detail, and what guarantees it provides.

Combining off-chain and on-chain channels is an idea presented in SDPP [84], but with one major difference. In SDPP, the on-chain channel is only used to record invoices and orders for data, and exchange cryptocurrencies between users, but not as a backup link. Using the blockchain as a backup link makes the publisher pay to deliver, by assumption, the message.

[84]: Radhakrishnan et al. (2018), 'Streaming Data Payment Protocol (SDPP) for the Internet of Things'

Assumptions

A Unidirectional Channel with On-Off Chain Proof of Delivery between a broker A and a broker B requires the following assumptions. Those assumptions should be verified to consider the channel *open*, and the way the two entities agree on those assumptions is not part of the protocol. In our architecture, this is done by SUPRA.

- Each broker has a pair of private-public cryptographic keys. Each broker is aware of the public key of the other broker. A broker can update its pair of keys, but if it does, the other broker has to be aware of it.
- Both entities have to be connected to the same blockchain, either by being part of it (*i.e.*, being a full node) or by being connected to another trusted full node (or set of full nodes) reliably. In other

words, we assume that each broker can receive events from the blockchain, for instance, every time a transaction associated with a specific ID or address is included in a block.

- Each broker can receive messages, of any type, on the blockchain. Here, we assume that each broker has a unique ID on the blockchain and there is a specific kind of transaction that can contain data of any type and is associated with a specific ID. So that when sending on-chain messages, a sender can create a transaction with the data and the corresponding broker ID. This message will be received by the broker eventually as it is supposed to be connected to the blockchain. In practice, this can also be implemented using a smart contract.
- The size of the messages is smaller than the maximum transaction size of the blockchain. This assumption can easily be removed by considering that only a fingerprint of the message is included in the transactions on the blockchain and that a publicly available distributed cloud storage is used to store the message. Doing so the messages are public and auditable, exactly like they are if they were included in the blockchain.
- Messages are weakly timestamped. This means that messages include the timestamp of the sender, but brokers will ignore a received message if the timestamp is in the future compared to its local clock or if the timestamps of consecutive messages are not increasing.
- The brokers agree on a value $T_{acknowledged}$, which represents the maximum acceptable duration between the first transmission of a message and its acknowledgment.
- $\Delta_{on-chain}$ is the maximum amount of time used by the blockchain network to add a transaction in a block. This strong assumption is common in blockchain-related works [39].

The Two Modes of Communication

Let m be the message broker A wants to send to broker B. The first mode of communication used by A is **the off-chain transmission**. A signs the concatenation of m with the signature of the previous message s_{previous} to obtain $s = \boldsymbol{sign}_A(m||s_{\mathit{previous}})$. Then, A sends the message and the signatures $m||s_{\mathit{previous}}||s$ directly to B. When B receives the message $m||s_{\mathit{previous}}||s$, it signs an acknowledgment Ack_m and sends it to A. The acknowledgment is basically, the signature from B of the signature of the received message $\boldsymbol{sign}_B(s)$. A is allowed to send a new message before receiving the acknowledgment, but it has to store all the non-acknowledged messages in order to re-transmit them if needed. Each new message from A follows the same procedure.

From the way messages are linked by signature, an acknowledgment implicitly acknowledges all the previous messages. So if B receives several messages from A that are correctly linked together (*i.e.*, no message is missing), B can send an acknowledgment only for the last received message.

If A does not receive the acknowledgment from B before a certain amount of time has passed (due to a connectivity issue or because B does not send it), A can send again the same message to ask B to send an acknowledgment again. A is free to decide when it re-transmits the message, however, it needs a proof of delivery before $T_{acknowledged}$ after the first transmission. To ensure B receives the message before $T_{acknowledged}$, A can use the second transmission mode: the on-chain method, described later. The second method may take a time $\Delta_{on-chain}$ before a message is confirmed (where $\Delta_{on-chain}$ is the time for a transaction to be included in the blockchain used by the brokers). Hence, after a delay $T_{\mathit{off}-ack} = T_{acknowledged} - \Delta_{on-chain}$ from the first transmission of a message, if no acknowledgment is received, A uses the second method of transmission to ensure the correct delivery before $T_{acknowledged}$.

For an on-chain transmission, A sends the message and its signature $m||s_{\mathit{previous}}||s$ to the blockchain associated with the ID of destination B. Doing so, as soon as the message is included in a block in the blockchain, A knows that B is aware of the message, by assumption.

In the case, B does not receive a message m from A, and A does not send it on-chain, then B has no way to know that it missed the message until A sends another message. When B finally receives the next message $m'||s_m||s$ from A, it can verify whether the signature s_m equals the signature of the last received message. If it is not equal, then B knows that message m is missing. B knows that A is supposed to send the message again off-chain or on-chain, so it just waits. If B does not receive the message m after $T_{acknowledged}$, it knows that A did not follow the protocol.

Delivery Guarantees

Our protocol offers several delivery guarantees and generates proofs that can be shown publicly to prove to anyone that a broker did follow the protocol.

Proof of integrity and origin of data: In the case where an off-chain message is correctly received, the message $m||s_{\mathit{previous}}||s$ is a proof for B that A sent the message. If there is a connectivity issue and B does not receive the message directly from A, then A sent the message on-chain, so the proof is the same. Hence, our protocol verifies the non-repudiation property.

Proof of delivery: In the case where an off-chain message and its acknowledgment are correctly received, the acknowledgment from B is a proof, for A, that B correctly received the message. If there is a connectivity issue and A does not receive an acknowledgment from B, then the on-chain message of A containing the data is a proof that B received correctly the message, by assumption.

Proof of non-delivery: If B detects that a message is missing, and does not receive it (off-chain or on-chain) before a delay $T_{acknowledged}$, then the signature of the missing message can be used to prove that A did not deliver correctly the message. Indeed, the signature proves that A sent a message, and if B did not receive this message, A cannot provide a proof of delivery.

7.4 Conclusion

In this chapter, we have presented the distributed publish/subscribe architecture in which we will run our unidirectional channel with on-off chain proof of delivery. In the next chapters, we will explain how SUPRA sets up and use this channel between the brokers, and how it uses the proof to detect malicious brokers.

First version of SUPRA | 8

SUPRA is a distributed publish/subscribe protocol between brokers using the unidirectional channels with on-off chain proof of delivery. Brokers share data from their local publishers to other brokers having subscribers, and they use topics as a filter for data. The protocol describes how brokers can set up subscriptions and exchange chained messages. There is no restriction on which protocol brokers use between them and local subscribers and publishers.

We are using Universal Unique Identifiers (UUID) to identify brokers. The link between a UUID and the public key of a broker is made in a smart-contract. Each transaction is associated with a source and a destination UUID, and each broker (being either connected to a blockchain node or a blockchain node itself) can listen to given UUIDs and receive the associated transactions when they are included in a block.

SUPRA is divided into five modules:

1. Public key module: managing the public identities of entities.
2. Subscription module: handling subscriptions.
3. Publishing module: publishing data using our previously defined communication channel.
4. Trial module: detecting malicious brokers using the blockchain.
5. Fail-over module: retrieving connection state after a crash from a broker.

We assume that messages are transferred between the brokers in a partially synchronous model on unreliable links. In this chapter, we will explain how the first version of the protocol defines each module.

8.1 Communication example

In Figure 8.1, we present a communication example between two brokers. In this example, A is the broker to which a subscriber is directly connected and B is the broker to which the publisher is directly connected. After declaring their public keys on the blockchain, broker A retrieves the public key of B and sends a subscription demand to B. At the reception of the demand, B retrieves A's key on-chain and checks the signature of the message. Thereafter, signature checking is performed by A and B at each message reception. In this example, the first data published by B is received and acknowledged directly by A using the off-chain channel. The message containing the second data publication is lost but the message disappearance is detected only at the reception of the third data publication because A detects a missing signature in the chained messages. This missing message detection is explained in Chapter 7. Hence, A warns B of a missing message, the message is resent, and A acknowledges it. Notice that A acknowledges the third data and not the second one because acknowledging a message implicitly acknowledges all the previous messages. Another way to retrieve a missing message is

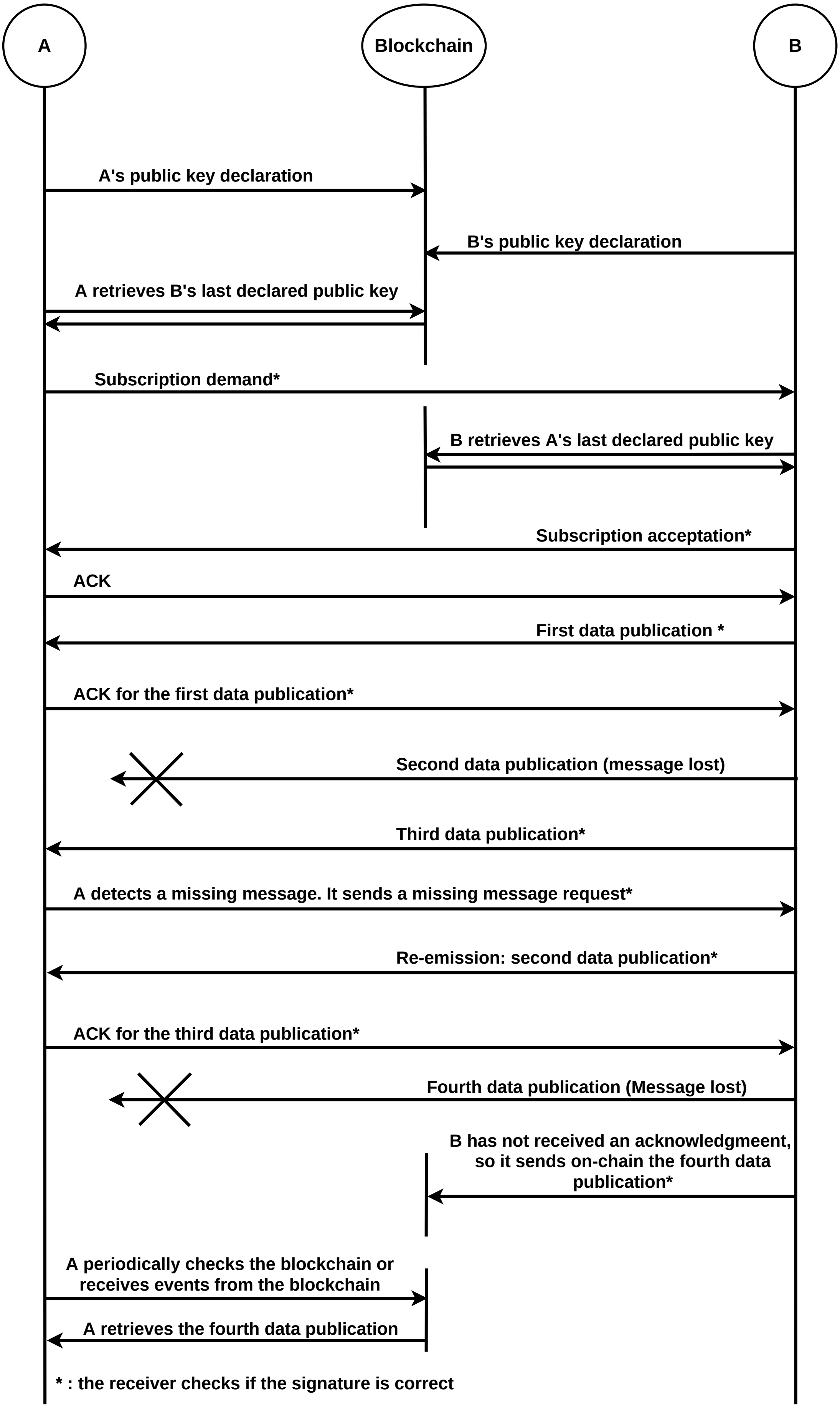

A
Blockchain
B
A's public key declaration
B's public key declaration
A retrieves B's last declared public key
Subscription demand*
B retrieves A's last declared public key
Subscription acceptation*
ACK
First data publication *
ACK for the first data publication*
Second data publication (message lost)
Third data publication*
A detects a missing message. It sends a missing message request*
Re-emission: second data publication*
ACK for the third data publication*
Fourth data publication (Message lost)
B has not received an acknowledgmeent, so it sends on-chain the fourth data publication*
A periodically checks the blockchain or receives events from the blockchain
A retrieves the fourth data publication
* : the receiver checks if the signature is correct

Figure 8.1: Message exchanges between two broker

shown in the fourth data publication. B resends the fourth publication on-chain because it does not receive the acknowledgment in time. A retrieves the message from the blockchain directly (either it receives the event from the blockchain node, or A requests periodically the blockchain for new messages). Observe that A does not acknowledge the last message. Indeed, an on-chain message is assumed to be always acknowledged (see Chapter 7). We will now provide in the next sections further details for this exchange between A and B.

8.2 Generic message format

All the messages of SUPRA are represented in Figure 8.2. They all have in common the following specifications. They start with two hexadecimal digits coded on one byte. The first digit is for the module and the second digit is for the type of message inside the module. We will write the message codes in this format: $X - Y$. Each code is associated with a specific action and format. Next to it, there is a timestamp. We use the UNIX timestamp format, which is an 8 bytes long unsigned integer with milliseconds precision. This timestamp adds unpredictability to messages. This property is important in a secure environment based on signatures. Indeed, if you have every possible output signed by someone, you can say everything you want on his behalf without his consent (*e.g.*, replay attacks). The timestamp prevents this type of behavior by creating a chronology between messages. This property is explained in further detail in Section 8.6. We will assume that two messages cannot have the same timestamp. Next to the timestamp, the message has the UUID of the destination. The last part of the message is a signature to prove the identity of the author. We use ED25519 as a signature protocol, where signatures are 64 bytes long and public keys are 32 bytes long.

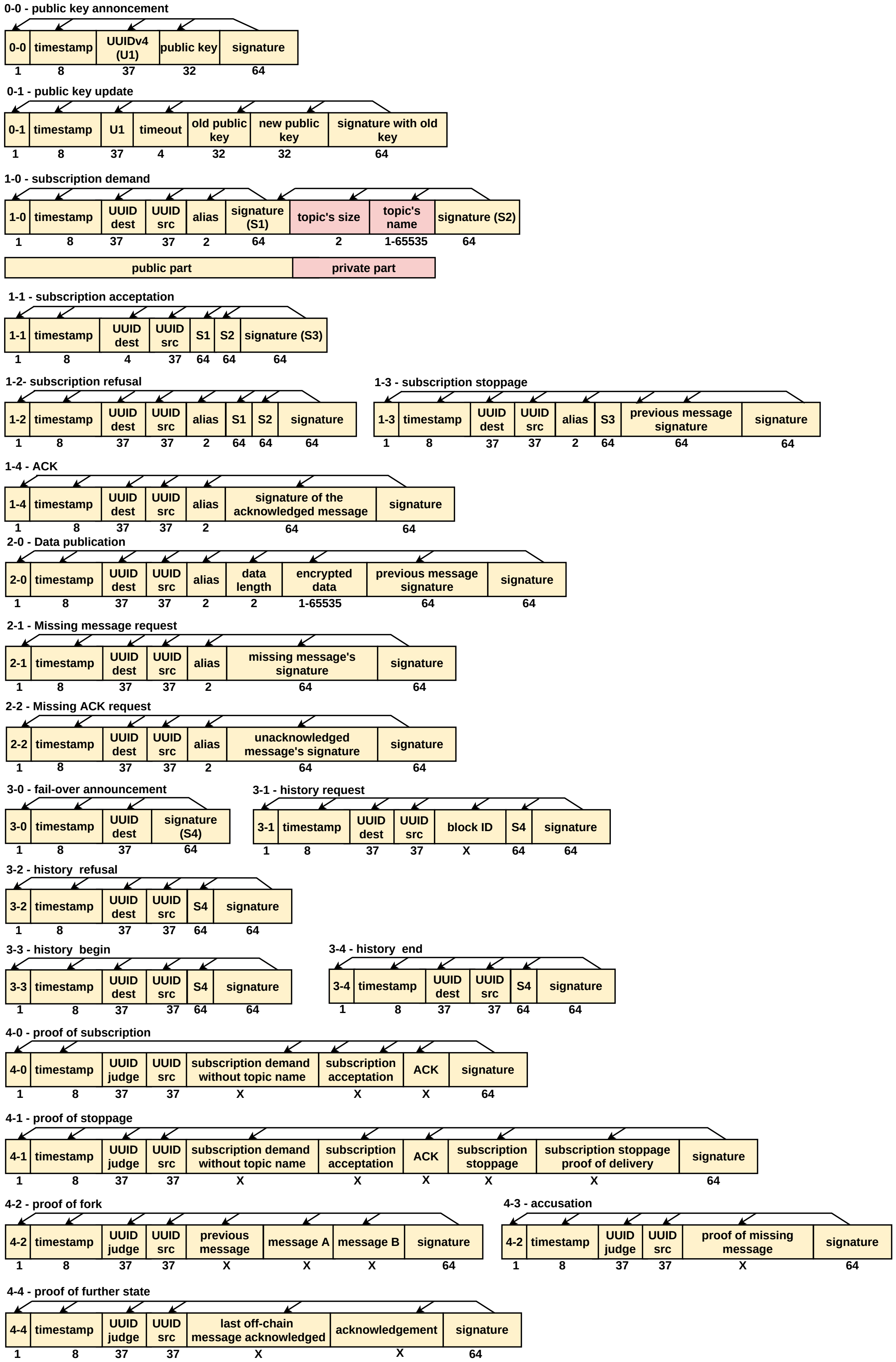

0-0 - public key annoncement
0-0 | timestamp | UUIDv4 (U1) | public key | signature
1 | 8 | 37 | 32 | 64
0-1 - public key update
0-1 | timestamp | U1 | timeout | old public key | new public key | signature with old key
1 | 8 | 37 | 4 | 32 | 32 | 64
1-0 - subscription demand
1-0 | timestamp | UUID dest | UUID src | alias | signature (S1) | topic's size | topic's name | signature (S2)
1 | 8 | 37 | 37 | 2 | 64 | 2 | 1-65535 | 64
public part
private part
1-1 - subscription acceptation
1-1 | timestamp | UUID dest | UUID src | S1 | S2 | signature (S3)
1 | 8 | 4 | 37 | 64 | 64 | 64
1-2- subscription refusal
1-2 | timestamp | UUID dest | UUID src | alias | S1 | S2 | signature
1 | 8 | 37 | 37 | 2 | 64 | 64 | 64
1-3 - subscription stoppage
1-3 | timestamp | UUID dest | UUID src | alias | S3 | previous message signature | signature
1 | 8 | 37 | 37 | 2 | 64 | 64 | 64
1-4 - ACK
1-4 | timestamp | UUID dest | UUID src | alias | signature of the acknowledged message | signature
1 | 8 | 37 | 37 | 2 | 64 | 64
2-0 - Data publication
2-0 | timestamp | UUID dest | UUID src | alias | data length | encrypted data | previous message signature | signature
1 | 8 | 37 | 37 | 2 | 2 | 1-65535 | 64 | 64
2-1 - Missing message request
2-1 | timestamp | UUID dest | UUID src | alias | missing message's signature | signature
1 | 8 | 37 | 37 | 2 | 64 | 64
2-2 - Missing ACK request
2-2 | timestamp | UUID dest | UUID src | alias | unacknowledged message's signature | signature
1 | 8 | 37 | 37 | 2 | 64 | 64
3-0 - fail-over announcement
3-0 | timestamp | UUID dest | signature (S4)
1 | 8 | 37 | 64
3-1 - history request
3-1 | timestamp | UUID dest | UUID src | block ID | S4 | signature
1 | 8 | 37 | 37 | X | 64 | 64
3-2 - history refusal
3-2 | timestamp | UUID dest | UUID src | S4 | signature
1 | 8 | 37 | 37 | 64 | 64
3-3 - history begin
3-3 | timestamp | UUID dest | UUID src | S4 | signature
1 | 8 | 37 | 37 | 64 | 64
3-4 - history end
3-4 | timestamp | UUID dest | UUID src | S4 | signature
1 | 8 | 37 | 37 | 64 | 64
4-0 - proof of subscription
4-0 | timestamp | UUID judge | UUID src | subscription demand without topic name | subscription acceptation | ACK | signature
1 | 8 | 37 | 37 | X | X | X | 64
4-1 - proof of stoppage
4-1 | timestamp | UUID judge | UUID src | subscription demand without topic name | subscription acceptation | ACK | subscription stoppage | subscription stoppage proof of delivery | signature
1 | 8 | 37 | 37 | X | X | X | X | X | 64
4-2 - proof of fork
4-2 | timestamp | UUID judge | UUID src | previous message | message A | message B | signature
1 | 8 | 37 | 37 | X | X | X | 64
4-3 - accusation
4-2 | timestamp | UUID judge | UUID src | proof of missing message | signature
1 | 8 | 37 | 37 | X | 64
4-4 - proof of further state
4-4 | timestamp | UUID judge | UUID src | last off-chain message acknowledged | acknowledgement | signature
1 | 8 | 37 | 37 | X | X | 64

Figure 8.2: SUPRA message format

8.3 Public key module

In this part, we explain how the public key module works and the message format it uses. The purpose of this module is to begin the setup of a channel described in Chapter 7 by sharing public keys between brokers. Messages in the public key modules are sent directly on the blockchain.

The first message of this module, code 0-0, is used to link the first public key with a specific UUID. This message is signed with the private key associated with the public key declared in the message to prove the ownership of the key pair.

Once a key is declared, it can be updated with the second message of the module, code 0-1. This message is useful for brokers when the integrity of the private key is at risk. This message contains a new public key, but it is signed with the old private key, to prove the ownership of the UUID. The key switch takes effect immediately after reaching a timeout, which is a timestamp on 4 bytes inside the update message.

By doing the key declaration and the key update on-chain, all brokers can retrieve the history of public keys linked with a specific UUID. Since all SUPRA's messages are timestamped, this history allows anybody to select the right public key to verify the message's signature. It creates traceability on the author's identity and it is further used by the trial module in Section 8.6.

8.4 Subscription module

We now explain how the subscriptions are done by explaining the two parts of subscriptions, the subscription demand, and the subscription stoppage.

8.4.1 Subscription demand

Brokers set up the subscription between brokers with a subscription demand. This demand is a triple handshake done off-chain between the two brokers. Figure 8.3 gives an overview of this handshake.

Indeed, as assumed earlier, the subscriber and the publisher are connected to two different brokers. This means that the subscriber cannot subscribe

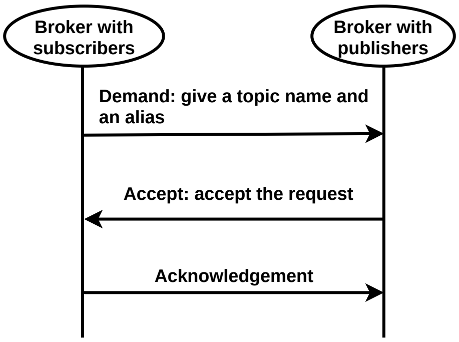

Broker with subscribers
Broker with publishers
Demand: give a topic name and an alias
Accept: accept the request
Acknowledgement

Figure 8.3: The triple handshake used to setup a subscription.

directly to the broker connected to the publisher. In that case, the broker connected to the subscriber has to subscribe itself, on the behalf of its subscriber, to the broker directly connected to the publisher. It is important to notice that if the broker has several subscribers interested in the same topic, it only has to subscribe once to the broker connected to the publisher. Once it retrieves the published data from the broker, it will forward it to its local subscribers.

The broker connected to the subscriber starts the handshake with a message with the code 1-0. In this message, it declares the topic name and an alias. The message is split into two parts. In the public part, the subscriber declares the alias and, in the private part, it declares the topic name. A second signature is used to link the topic name with the chosen alias.

Using an alias has two advantages. The first one is to identify each subscription by using less space than the topic name. If the broker has several subscriptions to the same broker at the same time, it needs information to know from which subscription a new data is. Adding the alias with the published data allows the users to identify the subscription by using a 2 bytes long value, instead of a topic name, which can be 65 535 bytes long. This optimization reduces the size of the exchanged messages. Notice that the same optimization is done in MQTT-SN [88], a version of the protocol aiming networks with no TCP/IP support. The second advantage is to prevent information leaks. Topic names can reveal private information on someone (Names, addresses), and to resolve issues between brokers we need to publicly reveal on-chain some messages. The public part of the message, with only the alias, is enough to prove the existence of a subscription, and it is better to reveal only a small random alias, instead of the whole topic name. This revealing process is explained in Section 8.6.

[88]: Hunkeler et al. (2008), 'MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks'

The second message of the handshake is the confirmation by the broker connected to the publisher. The message has the code 1-1. The third, and last, message of the handshake is an acknowledgment from the broker who started the handshake. To acknowledge off-chain a message, the receiver signs the signature of the acknowledged message. Once the handshake is done, data from the publisher can be shared between the brokers for the specified topic.

8.4.2 Subscription stoppage

To stop the subscription, either broker can send a stoppage message, because, for one broker, this method can be used because the subscriber is not interested anymore in the topic and, for the other broker, it can be used because the publisher stops the topic. One last reason can be, for the broker connected to the publisher, to stop the channel the other broker does not send enough acknowledgments. Indeed, the broker connected to the subscriber is not forced to send acknowledgments and it could wait that the other broker sends all the messages on-chain, and also pay fees. To prevent this behavior, the broker connected to the publisher can stop the subscription with the other broker at any time.

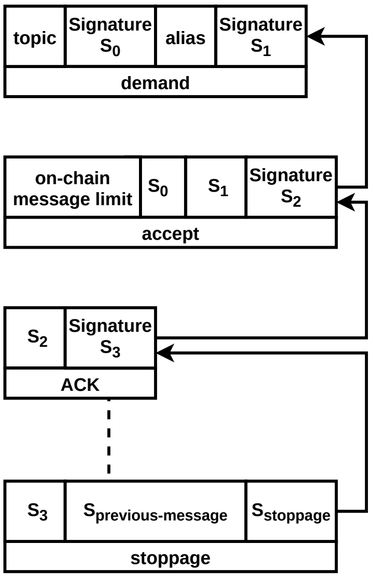


Figure 8.4: Chained messages in the subscription module messages.

The stoppage message, with the code 1-3, has to be acknowledged. When the message is explicitly acknowledged, or when the message is added in a block, the subscription is considered as ended.

As presented in Figure 8.2, which presents every message in SUPRA, each message of the handshake repeats the signature of the previous message. Figure 8.4 presents a simplified representation of these signature repetitions in the subscription module (*i.e.* timestamps, source UUID, and destination UUID are removed from the picture). The message used to stop the subscription repeats the signature of the last message of the triple handshake. So, regardless of how many messages are exchanged, once the channel is set up, it is always possible to link the stoppage message to a specific triple handshake.

8.5 Publishing module

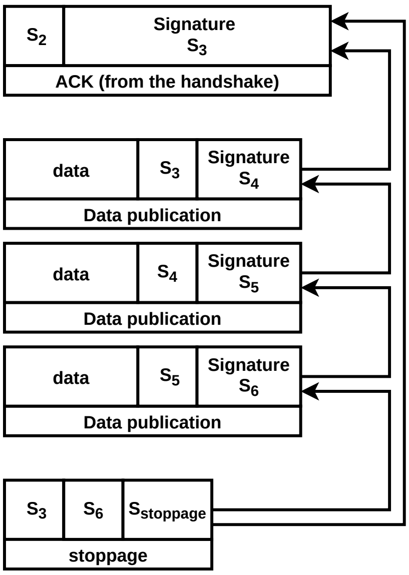


Figure 8.5: Chained messages in the publishing module

With the public key module, the brokers share public keys through the blockchain, and with the subscription module, they set up the channel by sharing a topic name, and an identifier for the subscription, the alias. Once the subscription is created, the publishing module describes how data from the publisher can be shared between the brokers, while respecting the properties of the unidirectional channel described in Chapter 7.

In a publish/subscribe protocol, the publisher wants to share information with the subscribers. To do so with SUPRA, the broker uses the message called data publication. This message has the code 2-0 and contains the data that the publisher wants to share. The broker will then share it with the other broker. Just like the messages in the hand-shake, the data publication repeats the signature of the previous message. The first data publication repeats the signature of the last message from the hand-shake. This repetition chains the messages together and can be observed in Figure 8.5. Chaining the messages allows the broker connected to the subscriber to detect missing messages, and to order messages if they arrive disordered because of network delay.

Messages from the publishing module go through a unidirectional channel with on-off chain proof of delivery, so the broker sending the message has to make sure that it is delivered before a timeout $T_{acknowledged}$, from the first message emission. This is possible because either the message is explicitly acknowledged by the other broker or because the message is sent in the blockchain network. To reduce the number of fees and to deliver as fast as possible messages, the broker connected to the publisher will first try to send the messages directly to the other broker. If the message is not acknowledged in time by the other broker, the message will be sent to the blockchain network. We assume that brokers are reliably connected to the blockchain network and that a transaction is added in a block in a maximum delay $\Delta_{on-chain}$, so a message is always available if it is sent to the blockchain network. The blockchain is used in this module as a backup channel with a full guarantee of delivery.

When the broker connected to the subscriber receives a new message from the other broker, it can detect whether there is a missing message, or if the chain of signature is broken. In case a message is missing, it can notify the broker connected to the publisher by sending the signature of the missing message using a warning message with code 2-1. This warning is not mandatory since messages have to be delivered before $T_{acknowledged}$, therefore missing messages will be sent in the ledger. We explain in Section 8.6 why is it important not to acknowledge a message before receiving all the previous messages and also how to detect brokers that do not respect the properties of the channel.

Also, brokers can ask to resent an acknowledgment to avoid sending a message on-chain. For the same reason as the previous warning, it is not mandatory to send this warning.

A message has to be acknowledged before $T_{acknowledged}$ and a transaction is added in a block, at most, in $\Delta_{on-chain}$, which means that if the acknowledgment is not received before $T_{acknowledged} - \Delta_{on-chain}$, the broker will send the message on-chain to ensure that the message is available before $T_{acknowledged}$. In that case, $T_{acknowledged}$ and $\Delta_{on-chain}$ have to be set based on the blockchain used in the system but also based on the delay in the network, because we want to avoid the acknowledgment of the broker connected to the subscriber reaches the other broker after $T_{acknowledged} - \Delta_{on-chain}$.

Published data is asymmetrically encrypted inside the message. The broker connected to the publisher uses the public key declared on-chain by the broker connected to the subscriber. This method avoids third parties to read published data, but it forces the broker to encrypt data for each broker. It means that if there are N brokers interested in the same topic, the broker connected to the publisher has to encrypt data N times. This problem can be resolved by adding in the protocol a message for the brokers to share a symmetric key, linked to the topic, through a Diffie-Hellman procedure with each other broker.

8.6 Trial module

The purpose of SUPRA is to have a publish/subscribe protocol that respects the properties of the channel presented in Chapter 7. The trial module allows brokers to detect brokers who do not respect the properties of the channel. To do so, they present exchanged messages to a smart contract. To prevent brokers from storing all the exchanged messages, before explaining how messages are compared, we will explain how they can store the right amount of messages.

8.6.1 Message conservation

The trial is based on comparisons between exchanged messages, so brokers need to retain messages for a certain amount of time. Not storing messages removes the chance for a broker to defend itself in case of a false accusation.

The triple handshake exchanged to set up the subscription has to be kept during the whole lifetime of the subscription, because the broker can prove the existence of the subscription with these messages.

With SUPRA, the broker connected to the publisher sent chained messages to a broker having subscribers. These messages can be acknowledged explicitly, with an acknowledgment from the brokers having subscribers, or implicitly when the message is sent on-chain. While a message is unacknowledged, it has to be stored, because the message may have to be resent. When messages are acknowledged, the two brokers just have to store the last messages explicitly acknowledged by the broker having subscribers and the acknowledgment. This message represents the last valid signature that the broker connected to the subscribers is explicitly informed.

After the stoppage of a subscription, messages can be deleted after a timeout T_{limit}. We will see how this timeout is defined in the next sub-section.

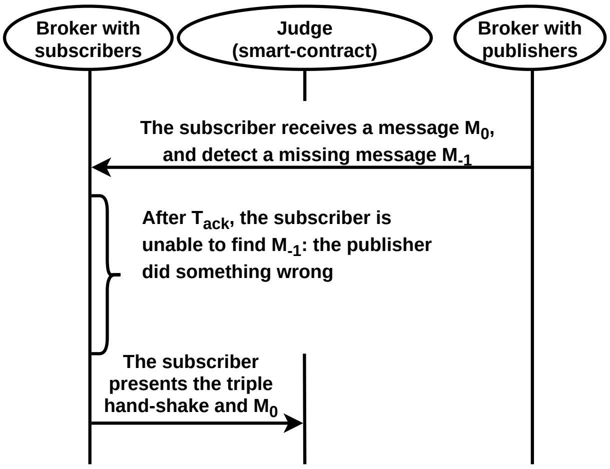

Broker with subscribers
Judge (smart-contract)
Broker with publishers
The subscriber receives a message M_0, and detect a missing message M_{-1}
After T_{ack}, the subscriber is unable to find M_{-1}: the publisher did something wrong
The subscriber presents the triple hand-shake and M_0

Figure 8.6: Declaring a misbehaving broker.

8.6.2 Trial process

To resolve issues between brokers, we will use a program with a trusted execution called the judge. To do so, the judge can be a smart contract inside the blockchain, because the execution of a smart contract can be verified by every user. T_{limit}, $T_{acknowledged}$, and $\Delta_{on-chain}$ can be declared inside the smart contract, and be available to all brokers before starting a subscription. We will assume that the judge is already running and that every broker knows how to contact it.

By detecting malicious brokers, honest brokers can reduce the risk of sending on-chain messages and paying fees. Also, since we use a smart contract, all accusations can be checked by all brokers. This accusation history can be used to compare two brokers who provide the same data, for instance, brokers could subscribe to the brokers who always deliver in time the messages, and one should avoid interacting with a broker that is known to send wrong accusations.

If a broker did not respect the specification of the channel, it means that a message was not delivered to a broker with subscribers before $T_{acknowledged}$. Messages are supposed to be delivered by off-chain or on-chain means before this timeout.

Figure 8.6 sums up the steps to detect a misbehaving broker in the system. When the broker connected to the subscribers receives a message, through the on-chain or the off-chain channel, it can detect a missing message, because there is a signature missing in the chain of signatures. The off-chain channel is unreliable and messages can be lost or disordered, so sometimes a message is considered missing while it is actually delayed. That is why the broker has to wait for $T_{acknowledged}$ before contacting the judge. If the broker connected to the publisher is honest, the missing message is delivered before $T_{acknowledged}$, because the broker resends the missing message off-chain or directly on-chain.

At the expiration of the timeout $T_{acknowledged}$, if the broker connected to the subscribers still cannot find the missing message, it means that the other broker did not respect the specification of the channel, since the message should, at least, be available on-chain. When the broker detects this misbehavior, it can present to the judge the triple-handshake used

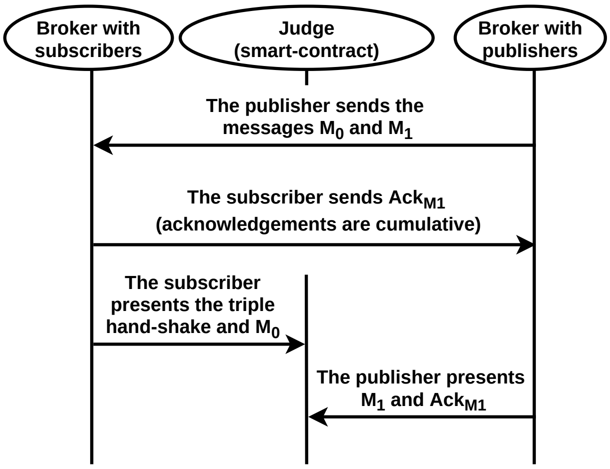

Broker with subscribers
Judge (smart-contract)
Broker with publishers
The publisher sends the messages M_0 and M_1
The subscriber sends Ack_{M1} (acknowledgements are cumulative)
The subscriber presents the triple hand-shake and M_0
The publisher presents M_1 and Ack_{M1}

Figure 8.7: Defending against a wrong accusation.

to set up the subscription and the message used to detect the missing message (M_0 on Figure 8.6). The judge can check the validity of the accusation by checking the signatures in the messages and by observing that the missing message is not in a previous block. In this case, the broker connected to the publisher will be unable to present a proof of delivery for the missing message, because the missing message is not in any block and was never acknowledged by the other broker.

We can see why it is important for the broker connected to the subscribers to not acknowledge the message used to detect a missing message until the missing message is retrieved. Since acknowledgments are cumulative, if it acknowledges this message, it also acknowledges the missing message. Thus creating a proof of delivery for the broker connected to the publisher for a message that is never delivered.

If the broker connected to the subscribers wrongly accuses the other broker of not respecting the specifications of the channel, the other broker is always able to defend itself, as long as it respects the message conservation rules presented in Section 8.6.1. This process is explained in Figure 8.7. Wrong accusations can only be done for messages exchanged off-chain, because the smart contract can easily verify if an on-chain message was added in time in a block. If the broker connected to the publisher respects the specifications of the channel, all messages are acknowledged explicitly, by the other broker, or by assumption, by sending the message on-chain. Since a wrong accusation can only target a message exchanged off-chain, then, if the broker connected to the publisher can present the last message acknowledged off-chain, it is enough to prove that the accusation is wrong. By comparing the timestamp inside the message used by the accusation and the one inside the message presented by the broker connected to the publisher, the judge can deduce that the message presented by the broker, and acknowledged by the other broker, was sent after the message inside the accusation. Since acknowledgments are cumulative, presenting a most recent message proves to the judge that the broker connected to the subscribers received the presumed missing message.

The broker connected to the publisher can also break the specifications of the channel by creating two different messages with the same previous message (recall that the chain of signatures should not have forks). If the other broker detects this error, it can present the two messages to the judge. It is impossible for the broker connected to the publisher to defend itself against such accusations.

8.7 Network issue and fail-over

The off-chain channel between brokers is unreliable, which means that messages that go through this link can be delayed or dropped. These issues are handled by SUPRA with the timeout defined in the protocol and the on-chain channel. Indeed, we assume that the brokers are reliably connected to the blockchain network and that $\Delta_{on-chain}$ is the maximum amount of time to add a transaction in a block. This means that no matter the quality of the link between the broker, the broker connected to the publisher can always make the data available for the broker connected to the subscribers by sending it on-chain. If the quality of the link between the brokers is not sufficient to do the triple hand-shake, then brokers have to retry the setup phase of the subscription.

SUPRA has a fail-over module to allow brokers to retrieve the aliases and the last sequence signatures of their current subscriptions when they crash. This process is presented in Figure 8.8. After a broker crashes, Broker A on Figure 8.8, when it is back online, it needs first to declare the crash on-chain. This is done with a message called fail-over announcement, with the code 3-0. Then, it can notify the other brokers of the crash, Broker B on Figure 8.8, and request them to resend every stored message related to subscriptions between them. The request indicates the block where we can find the fail-over announcement. When the requested broker has checked if the announcement is present on-chain, it can accept or refuse the request. As presented in Section 8.6.1, brokers have to store, at least, the triple handshake, and the last messages acknowledged off-chain for every active subscription to defend themselves in case of delivery issues.

If the request is refused by the broker, then all the active subscriptions between the two brokers are stopped, because the crashed broker is not

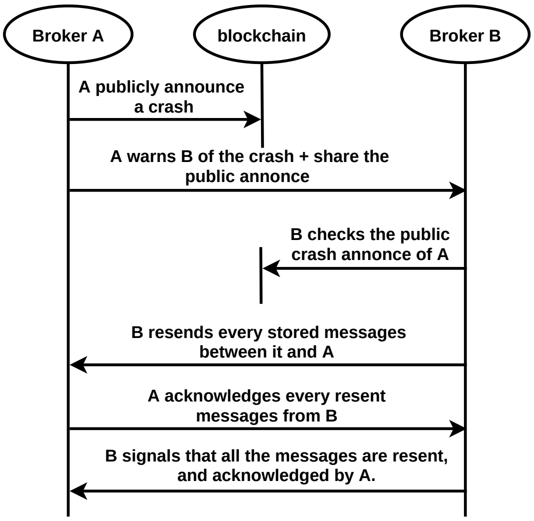

Broker A
blockchain
Broker B
A publicly announce a crash
A warns B of the crash + share the public annonce
B checks the public crash annonce of A
B resends every stored messages between it and A
A acknowledges every resent messages from B
B signals that all the messages are resent, and acknowledged by A.

Figure 8.8: Message exchange for crash recovery.

able to verify the chain of signatures.

If the request is accepted, then the broker has to resent every stored message. This resend process has to be done off-chain, each message has to be acknowledged by the crashed broker. This is to prevent fee attacks. Indeed, if the stored messages can be sent on-chain, then a malicious broker could declare a false crash, and force the honest broker to resend all the stored messages on-chain, by not acknowledging these messages. The malicious broker would have to pay the fees for one message, the public announcement of the crash, but would have forced the honest broker to pay fees for every stored message.

The same fail-over announcement can be used to join several brokers. So a broker can contact all its known brokers and retrieve all its subscription states. On the other hand, the same fail-over announcement cannot be used for two different crashes. If a broker tries to do so, it will be easily detected, because the other broker already has received a history request with this fail-over announcement or because the timestamps do not match. For instance, subscriptions created after the fail-over announcement message cannot be used to recover the state of these subscriptions.

Declaring errors on the blockchain makes broker behaviors publicly auditable. Those behaviors can be used by other brokers to measure a broker's reliability. Brokers want to send as few transactions as possible in the ledger because they cost money and take more time to be delivered, so they prefer to cooperate with brokers that force the least number of on-chain messages.

8.8 Comparison with existing solutions

In this section, we compare SUPRA with two solutions: MQTT and Trinity, which is the first combination of MQTT and blockchain to our knowledge.

MQTT

We compare MQTT and SUPRA to verify that SUPRA removes the issues linked with having a central broker.

The two common ways to add security in MQTT are to use SSL/TSL encryption and to use an authentication system on the broker. It is enough to remove overhearing by a third party but not enough to remove the star topology with the central broker. This broker can be the source of trust issues if it is malicious. In SUPRA, brokers communicate directly with each other, to create the subscription and send data. It removes the central authority from the system.

Still, the biggest difference between MQTT and SUPRA is about the trust assumptions between the users. In MQTT, the publishers and the subscribers must trust the central broker, if it is not the case they need to connect themselves to another broker. In SUPRA, publishers and subscribers must trust their local broker, so, just like MQTT, if they do not trust the local broker they must connect themselves to another broker.

On the other hand, the brokers do not trust each other. Also, thanks to the help of the blockchain, brokers can come and go as they want in the network of brokers. Once the broker declared a public key in the ledger, it just has to open channels with brokers sharing or wanting interesting data to join the system, and it can stop these channels at any time to leave the system. This dynamic property is interesting to interconnect IoT networks.

Also, MQTT runs over TCP [89]. TCP gives guarantees on message delivery: messages are ordered and they reach the destination if it is available. MQTT could be used to interconnect brokers but the problem is that we cannot make the difference between an offline/unreachable broker and a malicious broker who does not send acknowledgments. The result is the same for the honest broker, it has to resend infinitely messages.

SUPRA also guarantees message ordering and message delivery with the help of the unidirectional channel with on-off chain proof of delivery, but it introduced a maximum delay $T_{acknowledged}$ to deliver a message. If the sender does not receive acknowledgments, the message will be sent to the ledger and malicious brokers will not be capable of denying this message.

The on-chain and off-chain communications allow SUPRA to have level 2 in MQTT's QoS classes *i.e.*, messages are delivered exactly once, because brokers will ignore re-transmissions of a previously received message, between untrusted brokers.

These differences between MQTT and SUPRA make SUPRA more secure and also increase its number of use-cases. SUPRA can be used when traceability is needed in data exchanges or where the central broker is replaced with a dynamic network of brokers.

Trinity

To compare Trinity [3] and SUPRA, we use the 19 distributed publish/subscribe protocol threats presented in [90]. These threats were used in [90] to compare several security solutions in publish/subscribe protocols. Each threat is presented by its ID in the survey, TX where $X \in \{1, 2, ..., 19\}$.

[90]: Uzunov (2016), 'A survey of security solutions for distributed publish/-subscribe systems'

Trinity and SUPRA are alike against several threats because these protocols share similar properties. Messages are signed, which prevent attacks based on data alterations or spoofing ($T4, T5, T6, T7, T10$) and there are mechanisms to prevent malformed messages ($T9$). In Trinity, it is the API used by the brokers and, in SUPRA, it is the judge. Malformed proofs are ignored by the judge. The blockchain also allows these protocols to handle crashed brokers because data lost during a crash are available on-chain after the crash recovery ($T18$).

In Trinity, all the data published on a blockchain is only accessible by the brokers. In SUPRA, we use a new communication paradigm to send data off-chain as much as possible. The blockchain is used to create trust in brokers' identities and as a backup communication link for data. Despite this difference between Trinity and SUPRA, both protocols are protected against unauthorized actors ($T11$) and data repudiation

($T17, T19$). However, there is a risk of eavesdropping ($T1, T2$) in Trinity, because data are not encrypted.

Trinity's blockchain network is only available for brokers, it supposes a minimal amount of trust between brokers, because of this trust, data are not encrypted. Thus, brokers have access to every published data, even data from a topic for which they have no subscribers. In SUPRA, data are encrypted and only the receiver can decrypt it, which removes eavesdropping. Brokers trust the blockchain and the protocol as a whole, but they do not trust other brokers.

Both protocols don't implement tools against flooding and denial of service-based attacks ($T14$, $T15$, $T16$, $T17$). SUPRA and Trinity also share a risk of subscription leak ($T12$), if a broker is hacked. These leaks will only concern the subscriptions of the hacked broker but not every subscription in the network. Trinity implements a QoS system on published data in [4], this feature is not present in SUPRA but can be added in future works.

[4]: Ramachandran et al. (2018), 'Trinity: A Distributed Publish/Subscribe Broker with Blockchain-based Immutability'

SUPRA and Trinity do not implement routing protocols (layer 3) in their specifications, so we think that it is not meaningful to compare them against route poisoning attacks ($T7$).

8.9 Proof of concept

During our presentation of SUPRA, we explained how brokers exchange messages to set up and use the communication channel and we also proposed a complete message format. To prove that our proposition is realistic we made a proof of concept compatible with Ethereum. The code of our proof of concept is available online [91].

Ethereum is a crypto-currency that allows the creation of smart contracts. The presence of smart contracts is mandatory for SUPRA, because we need smart contracts to implement the judge used to resolve conflicts. Also, Ethereum is well known in the literature, since several works use this blockchain for their implementation [3, 4, 81, 92].

[3]: Ramachandran et al. (2019), 'Trinity: A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence'
[4]: Ramachandran et al. (2018), 'Trinity: A Distributed Publish/Subscribe Broker with Blockchain-based Immutability'
[81]: Lv et al. (2019), 'An IOT-oriented privacy-preserving publish/subscribe model over blockchains'
[92]: Danish et al. (2019), 'A Lightweight Blockchain Based Two Factor Authentication Mechanism for LoRaWAN Join Procedure'

Our proof of concept allows users to put online several brokers and attach subscribers and publishers to them. For a simple setup, we can create two brokers, one publisher connected to the first broker publishing data in topic 1, and one subscriber connected to the second broker subscribed to topic 1 of the first broker. Figure 8.9 illustrates the architecture.

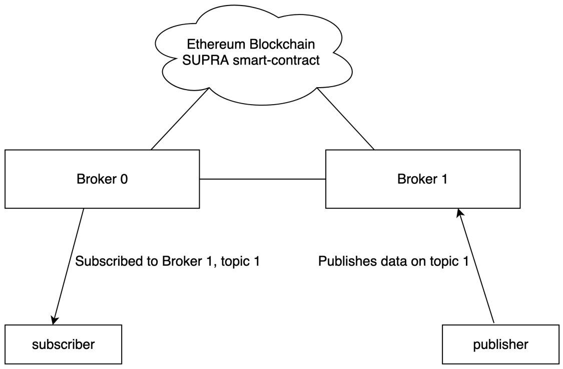


Figure 8.9: Architecture of our proof of concept

Once the system is set up, the subscriber can notify its local broker about the topic it wants. At the reception of this notification, the broker will create a channel between it and the broker connected to the publisher on the specified topic. When the publisher shares data with the topic, the data will go through the channel between its broker and the broker connected to the subscriber, and then reach the subscriber.

To simulate lost messages, data can be published with a specific flag. The broker connected to the subscriber ignores messages with the flag and it does not send an acknowledgment. To respect the rules of the channel, since the broker does not receive an acknowledgment in time, it sends the message to the Ethereum smart contract. When the message is added to the ledger, the other broker detects this message and forwards it to its local subscriber. The smart contract is written in Solidity. The publishers/subscribers, the brokers, and their interaction with the Ethereum blockchain are written in JavaScript.

Our smart contract also implements some functions for the judge that can be tested with unit tests. For instance, a broker A that receives a message M_2 can accuse another broker B not having sent it message M_1. In our unit tests, we created two scenarios, one where broker B defends itself by showing that M_1 is on-chain, and one where broker B defends itself by sending M_1 and its acknowledgment to the judge. In both cases, the judge accepts the proof and A pays a penalty to B. If no proof is provided, B pays a penalty to A.

The implementation of our broker is pretty simple. We did not execute extensive measurements on this proof of concept. Our purpose was only to prove the feasibility of data sharing between brokers by using the hybrid off-chain/on-chain method. This proves that it is possible to interconnect untrusted brokers, and so share data in a publish/subscribe manner. In practice, an MQTT broker can be used for communications between the broker and the local subscribers/publishers. SUPRA is only used for communications between brokers. This way, we can make assumptions about the performance of such types of communications. When data goes through the off-chain channel, the performance is similar to MQTT, if we put aside the signature checking process. On the other hand, messages that go through the on-chain channel are slower than messages send with MQTT, because users have to wait for a block with the messages in it, but this occurs only if brokers are disconnected or malicious.

8.10 Conclusion

SUPRA is a publish/subscribe protocol aiming at communication between untrusted brokers. To do so, the protocol ensures that the brokers connected to subscribers are sure to receive before a delay $T_{acknowledged}$ the messages from the broker connected to publishers. Also, we prove that it is always possible for an honest broker to prove the incorrect behaviors of a malicious broker.

In the next chapters, we will present two extensions for SUPRA. Those extensions correct two issues of the first version of the protocol while keeping the security properties.

Signature reduction | 9

In our first version of SUPRA, we focused on creating a communication channel between brokers presenting delivery guarantees for users. This channel places itself in a middle of a publish/subscribe architecture where brokers share data from publishers, but this first version of the protocol has one flaw. We realized that the protocol does not scale well with the number of brokers interested in the same topic.

In the first version of SUPRA, the communication between the brokers is client/server-based and it creates a scalability problem. If the broker connected to a publisher wants to share the same data with N brokers, it has to generate N different messages. It might not be a problem from computational resources point-of-view, since brokers are assumed to be unconstrained devices and they have to be reliably connected to the blockchain, but this issue has an impact on the number of messages send to the ledger. In this chapter, we explain the scalability problem of the first version of SUPRA and we explain how to resolve it.

9.1 Scalability issue

To simplify our presentation and resolution of the scalability issue present in the first version of SUPRA, in this chapter, we will named "publisher" a broker having at least one publisher and "subscriber" a broker having at least one subscriber. The communications that we are explaining are occurring between the brokers. The real publishers and subscribers are still loosely coupled and they do not know each other.

We presented in the previous chapter the 5 modules that composed SUPRA. Of these modules, the most important is the publishing module. Once the subscription is set up, the publisher, so the broker having at least one publisher, uses this module to send data to the subscribers, brokers having at least one subscriber for this topic. 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 chapter, we assume that there are N subscribers for the topic T. Each subscriber is identified by a unique number Sub, with $0 \leq Sub < N$. In Figure 9.1, we can observe the messages M_i that the publisher has to create to publish data D to the N subscribers in SUPRA. Each 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 T. This value is used to identify multiple subscriptions between the same publisher and the same subscriber without using the topic name T, just like port numbers.
- $Enc_{Sub}(D)$: data D encrypted with the subscriber public key.

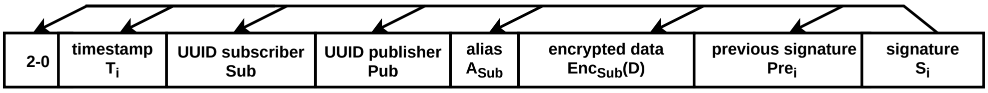


Figure 9.1: Data publication for the subscriber i

- 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 versions of M_i to share the same data D.

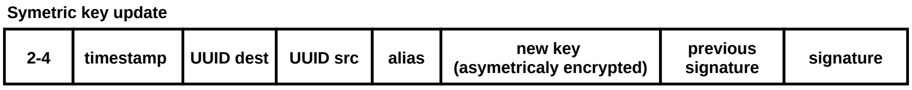
To avoid unauthorized entity to have access to D, the publisher encrypts data. In the first version of SUPRA, the publisher uses the public key of Sub to encrypt D. Each subscriber links a public key to its identifier in the distributed ledger. The public key of each subscriber is unique. It forces the 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*, 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 T, and by extension the value Pre_i. Since there are N versions of the message M_i, the 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 to 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.

* 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 $N \geq 302$.



Symetric key update

2-4	timestamp	UUID dest	UUID src	alias	new key (asymetricaly encrypted)	previous signature	signature

Figure 9.2: Symmetric key sharing message in SUPRA

9.2 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 9.3. 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.

9.2.1 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 Section 9.3, 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 9.2, 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 to as A_{Pub} the alias selected by the publisher for the topic T.

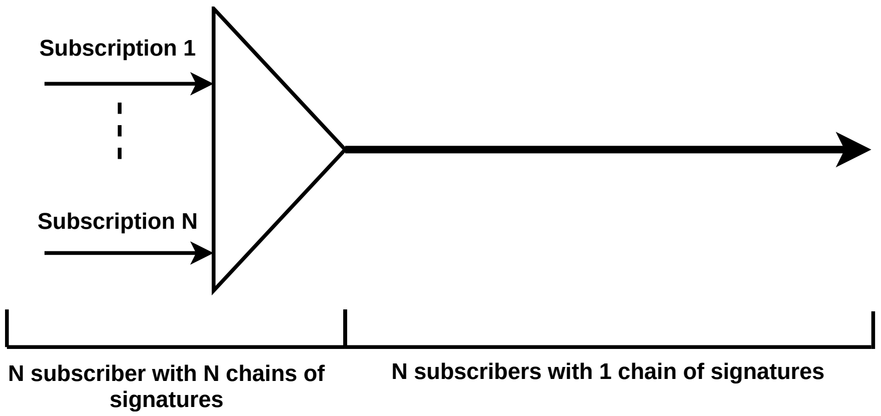

Subscription 1
Subscription N
N subscriber with N chains of signatures
N subscribers with 1 chain of signatures

Figure 9.3: Schematic representation of signature synchronisation.

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 versions 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

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

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 9.4. 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 9.5. 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,

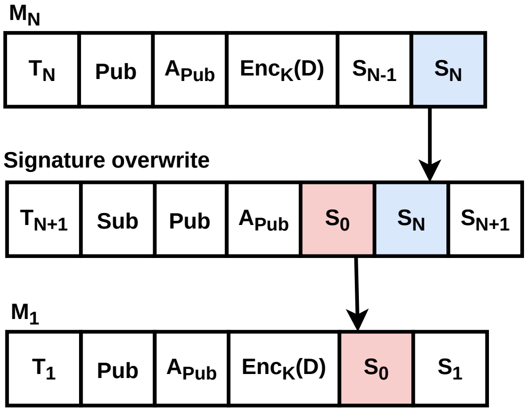


Figure 9.4: Example of signature overwriting

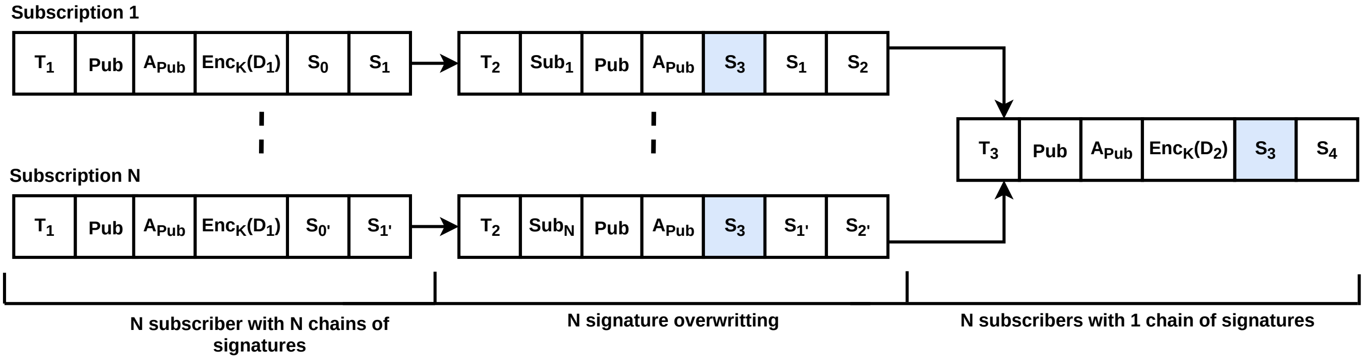


Figure 9.5: Synchronisation between several subscriptions.

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.

9.3 Security

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

9.3.1 Reuse acknowledgements

In SUPRA, messages are acknowledged explicitly by the subscriber or implicitly when they are added to a block. If the subscriber explicitly acknowledges the message, it signs the signature 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 9.3.1 *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 $S_{k'}$ □

9.3.2 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 unsubscription. 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.

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 9.3.2 *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. □

Preventing false accusation

Lemma 9.3.3 *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. □

9.4 Conclusion

By changing the message format of the data publication and by adding new control messages, we reduced the number of signature operations by the broker connected to the publisher for sharing the same data to N brokers. Now, the broker executes 1 signature instead of N. By doing so, we reduce the energy required by the broker to execute the protocol and we reduce the number of messages on-chain when a network failure occurs, thus reducing the cost of communication.

Still, over time, the broker connected to the publisher will just lose money because some message will be sent in the ledger. For the broker connected to the publisher, SUPRA only adds constraints. The broker has to deliver data in time, otherwise, it will be considered as misbehaving, and pay some transaction fees. These constraints can prevent the usage of the protocol for real-life scenarios. In the next chapter, we will present a solution for this problem with our last extension for SUPRA: a payment system allowing the broker connected to the publisher to gain money for sharing data through the SUPRA channel.

Data payment extension 10

SUPRA requests the broker connected to the subscriber to pay some transaction fees, because some messages will be sent in the ledger to be considered delivered in time. On the other hand, this broker gets nothing in return for providing such a delivery service for the other brokers. For this reason, adding a data payment extension for SUPRA is interesting to create an incentive for the broker to share data with the protocol.

In Chapter 6, we presented several data payment protocols, some of them using the blockchain. The blockchain can be interesting to share IoT data because it allows 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 €.* 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 [7] the smallest denomination of an Ether is called the Wei, which is 10^{-18} Ether, and also has a value smaller than 0.01€.

With publish/subscribe communication, the current state of the art of the proposition with blockchain usage is either dependent on one blockchain implementation [82] or the payment system adds an extensive delay for data delivery [87]. We already explained in Chapter 7 that SUPRA is not blockchain dependent and can run on several blockchain implementations. In this chapter, we will present an extension for SUPRA which reduces the impact of the payment system on data delivery and presents security guarantees for the buyer and the vendor: the broker with a publisher will get its payment, the broker with the subscriber will receive data.

[82]: Bu et al. (2019), 'HyperPubSub: Blockchain based publish/subscribe'

[87]: Ramachandran et al. (2019), 'Publish-pay-subscribe protocol for payment-driven edge computing'

The Lightning channels presented in Chapter 3 could be extended 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 requires complex secret exchanges between participants. Moreover, LN would require each broker to keep all the revoked transactions. Our solution achieves 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. That is why our proposition is secret-less.

10.1 Secret-less secured payment system

In this chapter, 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

* As of July 2022, 1 Satoshi = 0.00021 €

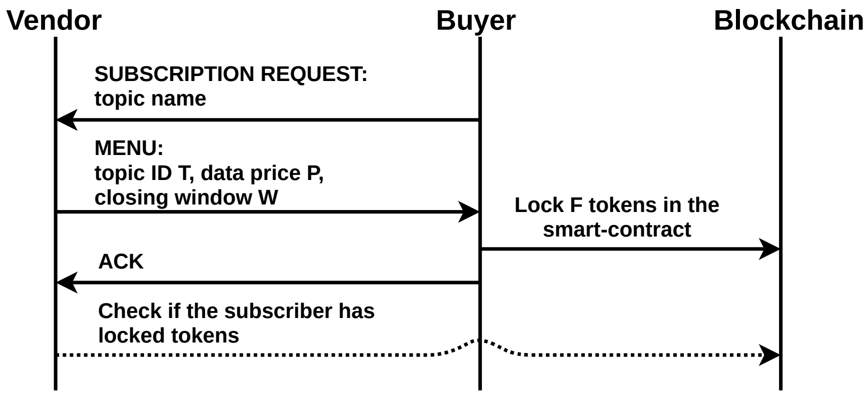


Figure 10.1: The publisher and the subscriber open a channel

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.

10.1.1 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 10.1. 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 chapter, we will assume that each data have a constant price P.

The closing window defines how much 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 10.2

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 10.1, 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.

10.1.2 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.

Payment using data acknowledgments

SUPRA acknowledgments contain the A's ID, and also the topic ID. For our payment system, we add a new field in the acknowledgments: the current cost C of the subscription.

As presented in Figure 10.2, 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 10.2, 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.

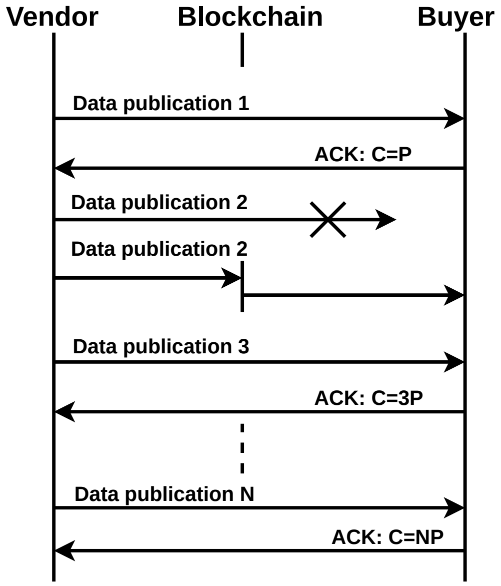


Figure 10.2: Payment promises through acknowledgments

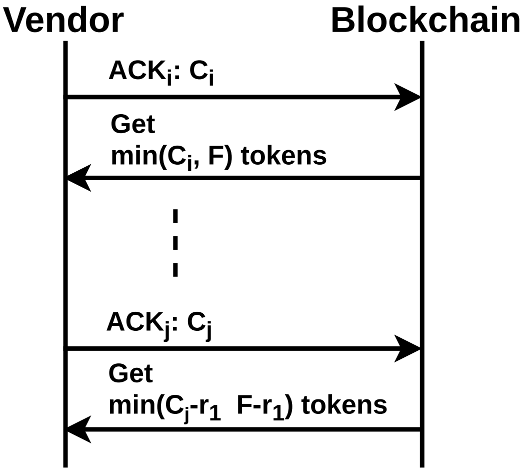


Figure 10.3: The publisher claims two times tokens from the contract

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 10.3. For instance, it can claim its token after i_1 messages, after i_2 messages, $\ldots$, 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 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.

Payment using on-chain messages

As we said previously messages can be lost when users use the off-chain channel. This 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 in the ledger, and the handshake used 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 10.2, we define this delay and explain how the smart-contract can detect such behaviors.

10.1.3 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.

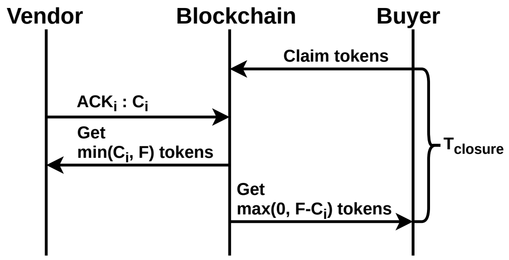


Figure 10.4: The subscriber reclaims its remaining tokens

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.

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 Lemma 10.2.4) 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 10.2, 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 10.4. This can be used in case A leaves 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.

Table 10.1: Comparison between our solution and SDPP in terms of on-chain messages and payment guarantees

	number of on-chain messages	maximum number of unpaid data	maximum number of data paid in excess
SDPP	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

After a delay $T_{closure}$, B receives the remaining tokens in the contract. Once the smart contract unlocks 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 number of tokens the vendor can claim will be wrong.

10.2 Security

In this section, we prove that our payment system provides the properties of an atomic swap [39]. This means that, if some users are malicious, correct users do not end up worse off.

[39]: Herlihy (2018), 'Atomic cross-chain swaps'

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 the 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 Chapter 8 what happens in case of conflicts on this property. In this chapter, we focus on malicious behaviors in the data payment system.

Lemma 10.2.1 *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 the 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 the 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 are any, are timestamped, and the price evolution is indicated in the handshake.

Lemma 10.2.2 *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 10.1.2, 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 10.2.3 *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 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 10.2.4 *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. □

10.3 Reputation system

Using the blockchain to resolve conflicts also creates a review system on users' behaviors. In [93], 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 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. Also, it can observe how much time the broker has claimed or locked tokens and so deduce when the broker joined the system.

[93]: Park et al. (2018), 'Smart contract-based review system for an IoT data marketplace'

We explain in Chapter 4, that some reputation systems using the blockchain can be useless against Sybil attacks: if a malicious user has a bad reputation, it will register itself with a new identity and a neutral reputation. To resolve this issue, we use the locked token as a finite resource to prevent such behaviors. Malicious brokers with subscribers will always lose the locked tokens if they try to deviate from the protocol and if the other broker is honest. Over time, they will just lose money.

With this system, brokers with publishers will never lose tokens, since they never lock tokens. To correct this problem, we can for instance force all brokers to lock some tokens when they register their public keys in the smart-contract. If a user detects an incorrect behavior from one broker, it wins the locked tokens, and also the potential tokens locked for the payment system. Thus all brokers have an incentive from following the protocol and detecting malicious brokers.

Also, it can be interesting for users to remain in the network with a good reputation for a long period. For instance, a broker could choose to reduce the data price with a broker requesting regularly the same topic. Also, it is possible to implement a broker where some topics are available for all users, but some 'premium' topics will only be available for brokers with correct reputations and in the network for a defined time. For instance, two topics for the same kind of data but one has some noises in the measurement and not the other.

10.4 Comparison with other solutions

In this section we compare our solution with SDPP [84], which the closest solution offering similar guarantees. Table 10.1 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.

10.5 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, we need a data payment protocol.

To the best of our knowledge, our solution for SUPRA 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;
- the protocol can run with different blockchain implementations;
- 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.

To conclude those works on SUPRA, we create a publish/subscribe protocol to inter-connect untrusted brokers, thanks to the blockchain. This protocol ensures data delivery before a certain delay. As long as the blockchain network remains trustable, a honest broker will always be capable of proving on the ledger that a malicious broker did not respect the delay. Compared to other state of the art solutions, our propositions reduces the number of messages sent to the ledger which reduce the cost of using the blockchain. On top of this delivery system, we add a payment system to create an incentive for the brokers which share data.

In the next chapter, we will present another propositions made during the thesis. With SUPRA, we focus ourselves improving security with the blockchain in the infrastructure on which IoT applications are running. With our next proposition, we propose a new consensus algorithm that does not used computational power as voting power, and could reduce the energy consumption of a blockchain network. This improvement can be a first step for implementing blockchain directly on IoT devices.

Proof of interaction

Proof of Interaction 11

In Chapter 2, we presented the current challenges for DLT. One of this challenge is the usage of computational resources. Indeed, the PoW forces users to invest in computational resources because, the more values it can try for the nonce the higher the chance to find the new block and get the reward. Investing in computational resources prevents Sybil attacks. The node can own several public keys it the network, it computational power remains the same. On the other hand, it creates a race for computational power between users and today the energy consumption used by the nodes to find new blocks is comparable to the energy consumption of a country *. Also, the PoW is vulnerable of selfish mining. An attack where a node does not share the new block to have an edge on the next block and increasing its reward, thus wasting honest nodes' energy.

There have been many attempts to avoid using Proof-of-work based agreement, but usually adding other constraints [94] (*e.g.*, small number of nodes, hardware prerequisite, new security threats).

In this chapter, we propose to use a new client-puzzle called Proof-of-Interaction to define a new energy-efficient Blockchain protocol. The Proof-of-work [14] is a method initially intended for preventing spamming attacks. It was then used in the Bitcoin protocol [1] as a way to prove that a certain amount of time has passed between two consecutive blocks. Our concensus algorithm is based on a work presented just prior the publication of Bitcoin in 2008, by M. Abliz and T. Znati [5]. They proposed *A Guided Tour Puzzle for Denial of Service Prevention*, which is another spam protection algorithm. This mechanism has not yet been used in the Blockchain context, and is at the core of our new Proof-of-Interaction. The idea was that, when a user wants to access a resource in a server that is heavily requested, the server can ask the user to perform a tour of a given length in the network. This tour consists of accessing randomly a list of nodes, own by the same provider as the server. After the tour, a user can prove to the server that it has completed the task and can then retrieve the resource. The way we generate our tour in our Proof-of-Interaction is based on the same idea. We generalized the approach of M. Abliz and T. Znati to work with multiple participants, and we made the tour length variable.

[14]: Back (2002), 'Hashcash - A Denial of Service Counter-Measure'
[1]: Nakamoto (2008), *Bitcoin: A Peer-to-Peer Electronic Cash System*

[5]: Abliz et al. (2009), 'A guided tour puzzle for denial of service prevention'

First, we propose a better alternative to Proof-of-Work, called Proof-of-Interaction, which requires negligible computational power. Second, we show how it can be used to create an efficient Blockchain protocol that is resilient against selfish mining, but assumes for now that the network is known.

* As of July 2022, the energy consumption of the Bitcoin network is equal to the energy consumption of Argentina
https://digiconomist.net/bitcoin-energy-consumption/

11.1 Preliminaries

11.1.1 Model

The network, is a set $\mathcal{N}$ of n nodes that are completely connected. Each node has a pair of private and public cryptographic keys. Nodes are uniquely identified by their public keys (*i.e.*, the association between the public keys and the nodes is common knowledge). Each message is signed by its sender, and a node cannot fake a message signed by another (non-faulty) node.

We denote by $\texttt{sign}_u(m)$ the signature by node u of the message m, and $\texttt{verif}_u(s, m)$ the predicate that is true if and only if $s = \texttt{sign}_u(m)$. For now, we assume the signature algorithm is a deterministic one-way function that depends only on the message m and on the private key of u. This assumption might be very strong as, with common signature schemes, different signature could be generated for the same message, but there are ways to remove this assumption by using complex secret generation and disclosure schemes, not discussed in this chapter, so that each signature is in fact a deterministic one-way function. The function H is a cryptographic, one-way and collision resistant, hash function [95].

As for the Bitcoin protocol, we assume the communication is partially synchronous *i.e.*, there is a fixed, but unknown, upper bound Δ on the time for messages to be delivered.

The size of a set S is denoted with $|S|$.

11.1.2 Guided Tour

The guided tour defined by M. Abliz and T. Znati [5] can be summarized as follow. When a resource server is under DOS attack, it responds to a given request by a random seed hash h_0, a set S of n servers and a length L. The client has to solve a puzzle in order to complete its request to the resource server. To solve the puzzle, the client makes L requests to the servers in S in a specific order. The index, in S, of the first server to request is deduced from h_0. Let $i_0 \in [0, n-1]$ such that $i_0 \equiv h_0 \mod n$. Then, the client sends message h_0 to the i_0-th server in S. The server responds with hash h_1. Then then client computes $i_1 \in [0, n-1]$ such that $i_1 \equiv h_1 \mod n$, and sends message h_1 to the i_1-th server in S, and so on. This continues until hash h_L is obtained. h_L is a proof that the tour as been completed, and is sent to the resource server to obtain the requested resource. Thanks to a secret shared among all the servers, the resource server is able to check that hash h_L is indeed the expected proof for the initial seed h_0. This idea is interesting because the whole tour depends only on the initial value, and cannot be performed in parallel because each hash h_i cannot be found until h_{i-1} is known. We then present a naive approach on how it can be used as a distributed client-puzzle.

11.1.3 Naive Approach

We give here a naive approach on how asking participants to perform a tour in the network can be used as a leader election mechanism to elect the node responsible for appending the next block in a Blockchain.

When a node u_0 wants to append a block to the blockchain, it performs a random tour of length L in the network retrieving signatures of each participants it visits. The first node u_1 to visit is the hash of the last block $h_0 = last\_block\_hash$ of the blockchain modulo n (if we order nodes by their public keys, the node to visit is the i-th with $i = h_0 \mod n$). u_1 responds with the signature $s_1 = \texttt{sign}_{u_1}(h_0)$. The hash $h_1 = H(s_1)$, modulo n, gives the second node u_2 to visit, and so on. This idea is similar to the guided tour of M. Abliz and T. Znati [5], and here the whole tour depends only the hash of the last block. Given h_0, anyone can verify that the sequence of signatures $(s_1, s_2, \ldots, s_L)$ is a proof that the tour has been properly performed. If each node in the network performs a tour, the first node to complete its tour is elected broadcast its block, containing the proof, to the other nodes to announce it.

However, here, each node has to perform the same tour, which could be problematic. An easy fix is to select the first node to visit, not directly using the hash of the last block, but also based on the signature of the node initiating the tour, $h_0 = H(\texttt{sign}_{u_0}(last\_block\_hash))$. Now, given h_0, the sequence of signatures $(\texttt{sign}_{u_0}(h_0), s_1, s_2, \ldots, s_L)$ proves that the tour has been properly performed by node u_0. Each tour, performed by a given node, is unique, and a node cannot compute the sequence of signature other than by actually asking each node in the tour to sign a message. Indeed, the next hop of the tour depends on the current one.

Here, one can see that it could be a good idea to also make the tour dependent on the content of the block node u_0 is trying to append. Indeed, using only the last block to generate a new proof does not protect the content of the current block, *i.e.*, the same proof can be used to create two different blocks. To prevent this behavior, we can assume that $h_0 = H(\texttt{sign}_{u_0}(last\_block\_hash) \cdot M)$ ($\cdot$ being the concatenation operator) where M is a hash of the content of the block node u_0 is trying to append. In practice, it is the root of the Merkel tree containing all the transactions of the block. Here, the proof is dependent on the content of the block, which means that if the content of the block changes the whole proof needs to be computed again.

From there, we face another issue. Each node performs a tour of length L, so each participant will be elected roughly at the same time, creating a lots of forks. To avoid this, we can make the tour length variable. We found two ways to do so. The first one is not to decide on a length in advance, and perform the tour until the hash of k-th signature is smaller than a given target value, representing the difficulty of the proof. In this way, every interaction with another node during a tour can be seen as a tentative to find a good hash (like hashing a block with a given nonce in the PoW protocol). The target value can be selected so that the average length of the tour is predetermined. However doing so, since the proof does depend on the content of the block, u_0 can change the content of the block, by adding dummy transactions for instance, so that the tour stops after one hop[†]. The other way to make the tour length variable is to use a cryptographic random number generator, seeded with $\texttt{sign}_{u_0}(last\_block\_hash)$, to generate the length L. Doing so, the length depends only on u_0 and on the previous block. Then a tour of length L is performed as usual.

To complete the scheme, we add other information to the message sent to the visited node so that they can detect if we try to prove different blocks in parallel. We also make u_0 sign each response before computing the next hop, so that the tour must pass through u_0 after each visit. Finally, we will see why it is important to perform the tour, not using the entire network, but only a subset of it.

11.2 The Proof-of-Interaction

In this section we define the most important piece of our protocol, which is, how a given node of the network generates a proof of interaction. Then, we will see in the next section how this proof can be used as an election mechanism in our Blockchain protocol.

11.2.1 Algorithm Overview

We present here two important algorithms. One that generates a Proof-of-Interaction (PoI), and one that checks the validity of a given PoI.

Generating a Proof-of-Interaction. Consider we are a node $u_0 \in \mathcal{N}$ that wants to generate a PoI. Given a fixed *dependency* value denoted d, the user u_0 wants to prove a *message* denoted m. The user has no control over d but can chose any message to prove.

The signature by u_0 of the dependency d, denoted $s_0 = \texttt{sign}_{u_0}(d)$, is used to generate the subset S of $n_S = \min(20, n/2)$ nodes to interact with

$$S = \{S_0, S_2, \ldots, S_{n_S-1}\} = \texttt{createServices}\left(\mathcal{N}, s_0\right).$$

S is generated using the pseudo-random Algorithm `createServices`, and depends only on d and on u_0. From s_0, we also derive the length of the tour $L = \texttt{tourLength}(D, s_0)$, where D is a probabilistic distribution that corresponds to the difficulty parameter. `tourLength` is a random number generator, seeded with s_0 that generates a number according to D. Using D one can easily change the average length of the tour for instance.

Now u_0 has to visit randomly L nodes in S to complete the proof, as illustrated in Figure Figure 11.1. To know what is the first node u_1 we have to visit, we first hash the concatenation of s_0 with m to obtain $h_0 = H(s_0 \cdot m)$. This hash (modulo $|S|$) gives the index i in S of the node we have to visit, $i \equiv h_0 \mod |S|$. So we send the tuple

[†] There are some ways to limit this attack, but we believe it will remain an important attack vector

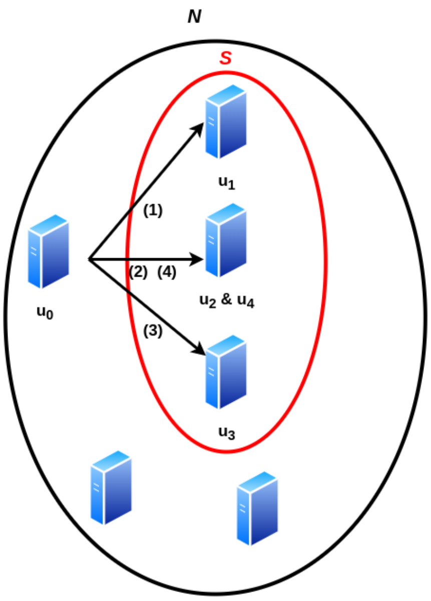


Figure 11.1: u_0 interacts randomly with a subset S of the nodes

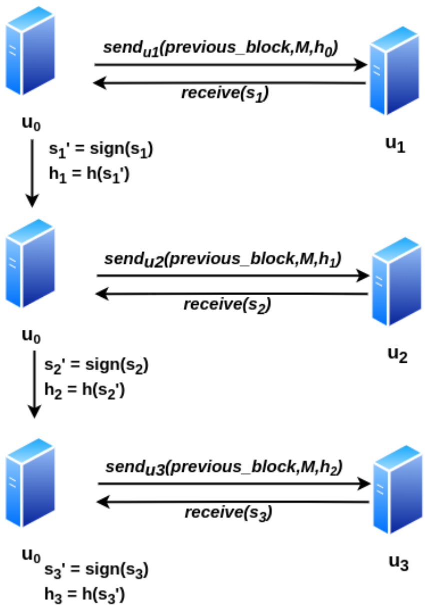


Figure 11.2: u_0 interacts with a sequence of nodes to construct a PoI. In this example, the dependency is the hash of the previous block.

(h_0, d, m) to node $u_1 = S_i$, which responds by signing the concatenation, $s_1 = \texttt{sign}_{u_1}(h_0 \cdot d \cdot m)$.

To know what is the second node u_2 we have to visit, we sign and hash the response from u_1 to obtain $h_1 = H(\texttt{sign}_{u_0}(s_1))$, so that $u_2 = S_j \in S$ with $j = h_1 \mod |S|$. Again, we send the tuple (h_1, d, m) to u_2, which responds by signing the concatenation, $s_2 = \texttt{sign}_{u_2}(h_1 \cdot d \cdot m)$. We sign and hash the response from u_1 to obtain $h_2 = H(\texttt{sign}_{u_0}(s_2))$ and find the next node we have to visit, and so on (see Figure Figure 11.2). This continues until we compute $\texttt{sign}_{u_0}(s_L)$, after the response of the L-th visited node.

The *Proof of Interaction* (PoI) with dependency d of message m by node u_0 and difficulty D is the sequence

$$(s_0, s_1, \texttt{sign}_{u_0}(s_1), s_2, \texttt{sign}_{u_0}(s_2), \ldots, s_L, \texttt{sign}_{u_0}(s_L)).$$

Checking a Proof-of-Interaction. To check if a PoI $(s_0, s_1, s'_1 \ldots, s_k, s'_k)$ from user u, is valid for message m, dependency d and difficulty D, one can first check if s_0 is a valid signature of d by u_0. If so, we can obtain the set $S = \texttt{createServices}(\mathcal{N}, s_0)$ of interacting nodes, the length $L = \texttt{tourLength}(D, s_0)$, and the hash $h_0 = H(s_0 \cdot m)$. From h_0 and S, we can compute what is the first node u_1 and check if s_1 is a valid signature from u_1 of $(h_0 \cdot d \cdot m)$, and if s'_1 is a valid signature of s_1 from u_0. Similarly, one can check all the signatures until s'_k. Finally, if all signatures are valid, and $k = L$, the PoI is valid.

11.2.2 Algorithm Details

The pseudo code of our algorithms are given below.

The algorithm createServices is straightforward. We assume that we have a random number generator (RNG) — defined by the protocol hence the same for all the nodes in the network — that we initialize with the given seed. The algorithm then shuffles the input array using the given random number generator. Finally, it simply returns the first n_S elements of the shuffled array.

Algorithm createServices: create a pseudo-random subset of nodes

```
Input: $N$, the set of nodes
       $h$, a seed
Output: $S$, a subset of nodes
$RNG.seed(h)$
$S \leftarrow$ shuffled$(N, RNG)$
$S \leftarrow S.slice(0, n_S)$
return $S$
```

The main part of the algorithm generatePoI consists in a loop, that performs the L interactions. The algorithm requires that each node in the network is executing the same algorithm (it can tolerates some faulty nodes, as explained later). The end of the algorithm shows what is executed when a node receives a message from another node. The procedure checkMessage may depends on what the PoI is used for. In our context, the procedure checks that the nodes that interacts with us does not try to create multiple PoI with different messages, and use the same dependency as everyone else. We will see in details in the next section why it is important.

The algorithm checkPoI that checks the validity of a PoI is checking that each signature from the proof is valid and respects the proof generation algorithm.

Proof-of-Interactions Properties. Now we show that the Proof-of-Interaction has several properties that are awaited by client-puzzle protocols [96].

Computation guarantee: The proof can only be generated by making each visited node sign a particular message in the correct order. The sequence of visited node depends only on the initiator node, on the dependency d, and on the message m, and cannot be known before completing the tour. Furthermore, a node knows the size of his tour before completing it, which means that the node knows before doing his tour how much messages it needs to exchange and how much signatures it will do to have a correct proof.

Non-parallelizability: A node cannot compute a valid PoI for a given dependency d and message m in parallel. Indeed, in order to know what is the node of the i-th interaction, we need to know h_{i-1}, hence we need to know s_{i-1}. s_{i-1} is a signature from u_{i-1}. So we can interact with u_i only after we receive the answer from u_{i-1} *i.e.*, interactions are sequential.

Granularity: The difficulty of our protocol is easily adjustable using the parameter D. The expected time to complete the proof is $2 \times mean(D) \times Com$ where Com is the average duration of a message transmission in the network, and $mean(D)$ is the mean of the distribution D.

```
Algorithm generatePoI: Program executed by $u_0$ to generate the PoI
Input: $d$, the dependency (hash of last block of the blockchain)
$m$, the message (root of the merkle tree of the new block)
$D$, difficulty of the PoI
$N$, the set of nodes in the network
Output: $P$, a list of signatures $\{s_0, s_1, s'_1, s_2, s'_2, \dots, s_k, s'_k\}$
$P \leftarrow [\,]$
$s_0 \leftarrow \texttt{sign}_{u_0}(d)$
$S \leftarrow \texttt{createServices}(N, s_0)$
$L \leftarrow \texttt{tourLength}(D, s_0)$
$P.append(s_0)$
$current\_hash \leftarrow H(s_0 \cdot m)$
for $L$ iterations do
    $next\_hop \leftarrow current\_hash \% |S|$
    $s \leftarrow send_{S_{next\_hop}}(current\_hash, d, m)$
    $P.append(s)$
    $s \leftarrow \texttt{sign}_{u_0}(s)$
    $P.append(s)$
    $current\_hash \leftarrow H(s)$
return $P$

When Receive $(h, d, m)$ from $u$ do
if checkMessage$(u, h, d, m)$ then
    Reply $\texttt{sign}_{u_0}(h \cdot d \cdot m)$
```

```
Algorithm checkMessage: Check the message received from node $u$
Input: $u$, the sender of the request
$h$, difficulty of the PoI
$d$, the dependency (hash of last block of the blockchain)
$m$, the message (root of the merkle tree of the new block)
Output: whether to accept or not the request
if $d$ is the hash of the latest block of one of the longest branches then
    if Received[$(u, d)$] exists and is not equal to $m$ then
        penalties ($u$)
        return false
    Received[$(u, d)$] = $m$
    return true
else
    if unknown $d$ then
        Ask block $d$
        return false
```

Efficiency: Our solution is efficient in terms of computation for all the participants. The generation of one PoI by one participant requires $mean(D)$ hashes and $mean(D)$ signatures in average for the initiator of the proof, and $mean(D)/n$ signatures in average for another node in the network. The verification requires $2D + 1$ signature verification and $mean(D)$ hashes in average. The size of the proof is also linear in the difficulty, as it contains $2mean(D) + 1$ signatures.

Algorithm checkPoI: Program executed by anyone to check the validity of a PoI

```
Input: $P$, a proof-of-interaction
$u$, creator of the proof
$d$, the dependency (hash of last block of the blockchain)
$m$, the message (root of the merkle tree of the new block)
$D$, difficulty of the PoI
$N$, the set of nodes in the network
Output: whether $P$ is a valid PoI or not
if not $\texttt{verif}_u$(P[0],d) then
    return $false$
$S \leftarrow \texttt{createServices}(N, P[0])$;
$L \leftarrow \texttt{tourLength}(D, P[0])$;
if $L * 2 + 1 \neq |P|$ then
    return $false$
$current\_hash \leftarrow H(P[0] \cdot m)$;
for $i = 0;\ i < L;\ i++$ do
    $next\_hop \leftarrow current\_hash\%|S|$;
    if not $\texttt{verif}_{S_{next\_hop}}(P[2*i+1], current\_hash \cdot d \cdot m)$ then
        return $false$
    if not $\texttt{verif}_u(P[2*i+2], P[2*i+1])$ then
        return $false$
    $current\_hash \leftarrow H(P[2*i+2])$;
return $true$
```

11.3 Blockchain Consensus Using PoI

In this section we detail how we can use the PoI mechanism to build a Blockchain protocol. The main idea is to replace, in the Bitcoin protocol, the Proof-of-work by the Proof-of-interactions, with some adjustments. We prove in the next section that it provides similar guarantees to the Bitcoin protocol.

Block Format. First, like in the Bitcoin protocol, transactions are stored in blocks that are chained together by including in each block, a field containing the hash of the previous block. In Bitcoin, a block includes a nonce field so that the hash of the block is smaller than a target value (hence proving that computational power has been used) whereas in our protocol, the block includes a proof of interaction where the dependency d is the hash of the previous block, and the message m is the root of the Merkel tree storing the transactions of the current block. Like for the transactions, the block header could contains only the hash of the PoI, and the full proof can be stored in the block data, along with the sequence of transactions.

Block Generation. Now we explain how the next block is appended in the blockchain. Like in Bitcoin, each participant gathers a set of transactions (not necessarily the same) and when the last block is received, wants to append a new block to the blockchain. To do so, each node tries to generate a PoI with the hash of the last block as dependency d, the root of the Merkel tree of the transactions of their own block as message m, and using the last block difficulty D. We assume the difficulty D is characterized by its mean value $mean(D)$, which is the number that is stored into the block. Like in Bitcoin, the difficulty can be adjusted every given period, depending on the time it takes to generate the last blocks.

Participants have no choice over d so the length of their tour, and the subset S of potential visited nodes is fixed for each participants (one can assume that it is a random subset). Each participant is trying to complete its PoI the fastest as possible, and the first one that completes it, has a valid block. The valid block is broadcasted into the network to announce to everyone that one have completed a PoI for its new block. When a node receives a block from another node, it checks if all the transactions are valid and then checks if the PoI is valid. If so, it appends the new block to its local blockchain and starts generating a PoI based on this new block.

First, one can see that this could lead to forks, exactly like in the Bitcoin protocol, where different part of the networks try to generate PoI with different dependencies. Thus, the protocol dictates that only one of the longest chain should be used as a dependency to generate a PoI. This is defined in the procedure `checkMessage`. When a node receives a message from another node, it first checks if the dependency matches the latest block of one of the longest chain. If not, the request is ignored.

Incentives. Like in Bitcoin, we give incentives to nodes that participate to the protocol. The block reward (that could be fixed, decreasing over time, or just contains the transactions fees) is evenly distributed among all the participants of the PoI of the block. This implies that, to maximize their gain, nodes should answer as fast as possible to all the requests from the other nodes currently generating their PoI, to increase their chance of being part of the winning block.

Also, it means that we do not want to answer a request for a node that is not up to date *i.e.*, that is generating a PoI for a block for which there is already a valid block on top, or for a block in a branch that is smaller than longest one.

Preventing Double-Touring Attacks. What prevents a node to try to generate several PoI using different variation of its block? If a node wants to maximize its gain (without even being malicious, but just rational) it can add dummy transactions to its current block to create several versions of it. Each version can be used to initiate the generation of a PoI using different tours. However, he has to send the message m every times he interacts with another node. If the length of the tour is long enough, the probability that two different tours intersect is very high. In other words, a node that receives two messages from the same node, with the same dependency d, but different values of m will raise the alarm. To prevent *double-touring*, it is easy to add an incentive to discourage nodes from generating several blocks linked to the same dependency. To do so, we assume each participant has locked a certain amount of money in the Blockchain, and if a node u has a proof that another node has created two different blocks with the same dependency (*i.e.*, previous block), then the node u can claim as reward the locked funds of the cheating node.

In addition, it can have other implications such as the exclusion of the network. We assume that the potential loss of being captured is greater than the gain (here the only gain would be to have a greater probability to append its own block).

Difficulty Adjustment. The difficulty could be adjusted exactly like in Bitcoin. The goal is to chose the difficulty so that the average time B to generate a block is fixed. Here, the difficulty parameter D gives a very precise way to obtain a delay B between blocks and to limit the probability of fork at the same time. If Com denotes the average duration of a transmission in the network, then we want the expected shortest tour length among the participants to be $\lceil B/Com \rceil$.

For instance, it is known that the average minimum of n independent random variables uniformly distributed on the interval (a, b) is

$$\frac{b + na}{n + 1}.$$

Thus, if D is the uniform distribution between 1 and $\lceil B/Com \rceil (n+1) - 1$, then the length of the shortest tour among all the participants will be $\lceil B/Com \rceil$ in average.

Every given period (*e.g.*, 2016 blocks as in Bitcoin), the difficulty could be adjusted using the duration of the last period (using the timestamps included in each block) to take into account the possible variation of Com, so that the average time to generate a block remains B.

Communication Complexity. A quick analysis shows that each node sends messages sequentially, one after receiving the answer of the other. At the same time, it answers to signature requests from the other nodes. In average, a node is part of n_S tours. Hence the average number of messages per unit of time is constant *i.e.*, $n_S + 1$ every Com. Then, the total amount of messages, per unit of time, in the whole network is linear in n.

11.4 Security

This section discusses about common security threats and how our PoI-based Blockchain handles them. We assume that honest nodes will always follow our algorithms but an attacker can have arbitrary behavior, while avoiding receiving any penalty (which could remove him from the network). We assume that an attacker can eavesdrop every messages exchange between two nodes but he can not change them. Also, assume that an attacker A cannot forge messages from another honest node B.

11.4.1 Crash Faults

A node crashes when it completely stops its execution. The main impact is that it does not respond to the sign requests of other nodes. This can be an issue because at each step of the PoI generation, the initiator node could wait forever the response of a crashed node. Crashed nodes are handled by the fact that a node only has to interact with a subset S of the whole network $\mathcal{N}$, computed using the service creation function, `createServices`. Hence, if a node crashes, only a fraction of the PoI that are being generated will be stuck waiting for it. All the nodes whose Service sets S do not contains crash faults are able to generate their PoI entirely. Since each set S is of size $n_S = \min(n/2, 20)$, we have that, if half of the nodes crash, the probability a given set S contains a crashed node is $1 - \left(\frac{1}{2}\right)^{n_S}$. So that the probability p that at least one set S contains only correct nodes is

$$p = 1 - \left(1 - \left(\frac{1}{2}\right)^{n_S}\right)^n$$

One can see that the probability p tends quickly (exponentially fast) to 1 as n tends to infinity. For small values of n, the probability is greater than a fixed non-null value. In the rare event that all the sets S contain at least a crashed node, then the protocol is stuck until some crashed nodes reboot and are accessible again.

Finally, we recall that honest nodes are incentivized to answer, because they get a reward when they are included in the next block's PoI. Hence, honest nodes will try be back again as fast as possible.

11.4.2 Selfish mining

Selfish mining [97] is an attack where a set of malicious nodes collude to waste honest nodes resources and get more reward. It works as follow. Once a malicious node finds a new block, it only shares it with the other malicious nodes. All malicious nodes will be working on a private chain without revealing their new block, so that honest nodes are working on a smaller public branch *i.e.*, honest nodes are wasting resources to find blocks on a useless branch. When honest nodes find a block, the malicious nodes might reveal some of their private blocks to discard honest blocks and get the rewards.

In Bitcoin, selfish mining is a real concern as attackers having any fraction of the whole computational power could successfully use this strategy [98].

[98]: Sapirshtein et al. (2016), 'Optimal selfish mining strategies in bitcoin'

Interestingly, our PoI-based Blockchain is less sensible to such attack. Our algorithm gives a protection by design. Indeed, when generating a PoI, a node has to ask to a lots of other nodes to sign messages containing the hash of the previous block, forcing it to reveal any private blocks. Other nodes in the network will request the missing block before accepting to sign the message. In other words, it is not possible to generate a PoI alone. Moreover, if a node is working on a branch that is smaller than the legitimate chain and ask for the signature of an honest node, the latter will tell the former to update its local Blockchain, thus preventing him from wasting resources.

11.4.3 Shared Mining

During the PoI, a node will most of the time be waiting for the signature of another node. So the network delay has the highest impact on the block creation time. To remove this delay, a set of malicious nodes can share theirs private keys between each other and try to create a set S where every nodes are malicious. If one malicious node of the pool succeeds, it can compute the proof locally without sending any messages. It will generate the PoI faster than honest nodes and have a high chance to win.

We defined earlier that each node of the network is known. Which mean that each node is a distinct entity. For this attack to succeed, entities need to share their private keys. This is a very risky move because once you give your private key to someone, he can create transactions in your name without your authorization. This risk alone should discourage honest nodes to do it, even if they want to maximize their gain.

We can still assume that a small number of malicious nodes do know each other and collude to perform this attack. We show now that this attack is hard to perform. S only depends on the previous block and on the identity of the initiator of the proof, so the nodes have no control over it. S consists of n_S nodes randomly selected among the network. So if there are F malicious friends on the network, there is on average the same fraction (n/F) of malicious friends in S as in $\mathcal{N}$. However, the probability for the tour to contains only malicious friend is very low. Indeed, with F malicious friends on the network, the probability that the entire tour consists of malicious friends is $(n/F)^{mean(D)}$ in average.

When a malicious node initiates a PoI for a given message, it can see whether the tour contains an honest node or not, so it might be tempted to change the content (by reordering the transaction or inserting dummy transactions) of its block until the tour contains only his malicious friends. However, even if there is a fraction $(n/F) = 0.1$ of malicious friends in the network (hence in S), and if $mean(D) = 100$, for instance, then the probability that a given tour contains only malicious friends is 10^{-100}. To find a tour with only malicious friends, an initiator would have to try in average 10^{100} different block content, which is not feasible in practice.

11.5 Conclusion and Possible Extensions

We have presented an new puzzle mechanism that requires negligible work from all the participants. It asks participants to gather sequentially a list of signatures from a subset of the network, forcing them to wait for the response of each visited node. This mechanism can be easily integrated into a Blockchain protocol, replacing the energy inefficient Proof-of-work. The resulting Blockchain protocol is efficient and more secure than the Bitcoin protocol as it is not subject to selfish-mining. Also, it does not have the security issues found in usual PoW replacements such as Proof-of-stack or Proof-of-elapsed time. However, it currently works only in networks where participants are known in advance. The design of our Blockchain protocol makes it easy to propose a possible extension to remove this assumption.

The easiest way to allow anyone to be able to create blocks, is to select as participants the n nodes that locked the highest amount of money. This technique is similar to several existing blockchain based on protocols that work only with known participants (such as Tendermint [99] using an extension of PBFT [100]) or where the nodes producing blocks are reduced for performance reasons (such as EOS [101] where 21 producer nodes are elected by votes from stakeholders). We believe a vote mechanism from stakeholders can elect the set of participants executing our protocol. The main advantage with our solution is that the number of participants can be very high, especially compared to previously mentioned protocols.

Conclusion 12

In this last chapter, we will summarize the propositions presented in this manuscript and present some possible future works.

12.1 Summary of our works

In this manuscript, we presented two contributions: SUPRA, a publish/-subscribe protocol and Proof of Interaction (PoI), a consensus algorithm.

12.1.1 SUPRA

SUPRA is the main contribution of this thesis. It is a publish/subscribe protocol built to create secured communications between untrusted brokers with the help of the blockchain. Publish/subscribe is a communication paradigm useful in IoT because there is a loose coupling between the publishers and the subscribers. It means that they do not need to know each other to share information. This property allows new publishers and subscribers to easily join or leave the system, which is interesting in IoT applications where the number of active devices is dynamic, and also the publishers and the subscriber does not need to be online at the same time to share data, which is interesting for constrained devices with batteries. To have this property, the brokers handle the complex operations of the paradigm. They store the subscriptions and forward data to the interested entities.

We observed in Chapter 5 that the blockchain can be used to increase the security between the brokers. Compared to other state-of-the-art solutions presented in this same chapter, SUPRA avoids as much as possible sending messages in the ledger, to reduce the cost of the communication and reduce the delay. Also, the protocol avoids using functions specific to one blockchain implementation. Allowing smart-contracts is the only requirement the protocol needs for the blockchain. In Chapter 7, we explained in more detail how we reduced the number of messages in the ledger and what are the requirements of the architecture.

In Chapter 7 we presented the important property of our communication channel: messages are delivered before a known delay because the broker connected to the publisher either has an explicit acknowledgment from the other broker or the message is added in a block and is assumed acknowledged. We presented the first version of the protocol in Chapter 8 where we showed how the brokers set up the communication channel so that the broker having publishers can share data with the broker having subscribers.

Also, we explained in this chapter how the broker having subscribers can detect if the broker having publishers does not respect $T_{acknowledged}$. This misbehaving can be proved to a smart-contract by presenting some exchanged messages between the brokers and all the other users can verify these messages using public information. We proved in our first version of the protocol that an honest broker cannot be proved misbehaving by a malicious broker and honest brokers can always prove that a malicious broker is misbehaving.

We presented in Chapter 9 and Chapter 10 two extensions for SUPRA. The first extension is an update to reduce the number of signatures executed by the broker with publishers to share data, and so the number of messages. We realized that, when the broker with publishers wanted to share the same data with several different brokers, it had to create one different message for each broker. Meaning that when the broker with publishers had to send a message in the ledger in order to respect $T_{acknowledged}$, different versions of a message sharing the same data ended up in the ledger, which is a waste of storage resources for the blockchain nodes and monetary resources for the broker. In the first extension, we explained this problem in more detail and how to correct it so that the broker with publishers only has to create 1 identical message for all the brokers interested in the same topic. Also, we proved that this modification does not change the security properties of SUPRA. Still, even with this extension, the broker with publishers will have to pay some fees and, over time, it just loses money. This issue can prevent the implementation of the protocol in real-life scenarios. For this reason, in Chapter 10, we presented a data payment extension for SUPRA. With this modification, the brokers have an incentive for sharing data with the protocol.

To summarize, we used the blockchain to create secured publish/subscribe communications between untrusted brokers. Those brokers can exchange data from publishers and have guarantees on data delivery or have proofs that something wrong happens. Thanks to a smart-contract, the incorrect behaviors can be presented to all the other users in the system. We tried to reduce as much as possible sending messages in the distributed ledger to increase the performance of SUPRA and reduce the cost of communications. Still, some fees will be paid by the brokers, and to encourage them to use our protocol we added a data payment extension. SUPRA increases the security of publish/subscribe communications and allows user to sell individually IoT data.

12.1.2 Proof-of-Interaction

By inspiring ourselves from a spam protection algorithm, we defined in Chapter 11 an energy-efficient puzzle-client that can be integrated into a blockchain protocol: the Proof of Interaction

Compared to the PoW, which is the most known consensus algorithm for blockchains, the PoI does not require extensive computational resources because the proof needed to validate the block is the list of signatures from other nodes in the network that can only be retrieved sequentially.

The interesting property of this signature retrieving process is that the node generating a proof is forced to share its previous block with the contacted nodes. Otherwise, the contacted node will refuse to participate in the proof, so the node will not be able to finish its proof. This sharing process prevents selfish-mining which is a known attack on PoW systems. Also, the PoI has linear complexity in the number of messages with the number of nodes on the network. Whereas other consensus algorithms relying on message exchanges between nodes, like PBFT, have quadratic complexity.

For IoT, we explained in Chapter 4 that the computational and storage resources required to execute blockchain clients prevent constrained devices from executing them. Thus, by reducing the computational resource required to take part in the consensus, the PoI can be a first step for reducing the requirements to be a full-node in a blockchain network and reducing the energy consumption.

12.2 Futur work and perspectives

We have several ideas for future works with SUPRA. The first short-term objective for SUPRA is to remove one assumption on the protocol. In Chapter 7, we explained that one of the assumptions used by the unidirectional channel with on-off proof of delivery is that the size of the messages is smaller than the maximum transaction size of the blockchain. With this assumption, published data are directly inside the SUPRA messages in an encrypted fashion, but it is not realistic if brokers share large data. For instance, in Ethereum, there is no limit on the transaction size. There is only a limit on the total number of gas in a block. In Ethereum, the transaction fees are the number of gas in the transaction multiplied by the price of one gas, and the larger the transaction, the more gas is needed, hence the more fees are paid by the user. So removing data from the on-chain messages can reduce the size of the message and the cost of using the blockchain.

To remove this assumption, we can replace the data shared inside the SUPRA with its fingerprint. Meaning that no matter the size of the exchanged data, the size of the SUPRA remains the same but it creates an issue with on-chain messages. In SUPRA, data is assumed acknowledged when added in a block because it is present in the on-chain message. If we remove data from the messages, brokers have to make available the published data elsewhere, in a storage place known by the other brokers. Adding data fingerprint in the ledger does not prove that data was available in time for the other brokers in this storage place. It means that to remove this assumption, we need to change the on-chain message format so that it includes a proof of the availability of the data in the shared storage place. We have some ideas on how to do this. For instance, IPFS [102] or other peer-to-peer file systems could be solutions to store published data, but this idea needs further investigation. An alternative is to inspire ourselves with the SLA monitoring serving presented in [64]. The judge smart-contract could be linked with a cloud storage place and when a broker has to send data on-chain, it sends the fingerprint in the ledger and the data in this storage place. Since smart-contracts cannot check information outside the blockchain, some users register themselves in the contract as inspectors and they check if data is truly available in the cloud storage system. This idea needs further work to explain how to prove data availability in the cloud storage system and how to encourage users to be inspectors.

[102]: Daniel et al. (2022), 'IPFS and friends: A qualitative comparison of next generation peer-to-peer data networks'

[64]: Zhou et al. (2019), 'A Blockchain based Witness Model for Trustworthy Cloud Service Level Agreement Enforcement'

For the PoI, our future works include removing the assumption on the fixed known size of the network. To verify the validity of a block made with a PoI, one must verify the seed used to build the set of nodes S and the length of the tour L. This set S is a subset of the network of nodes N. In our presentation of the PoI, we assumed that N is fixed and known, so nodes can easily check the validity of S, but it also means that no new nodes can join the system. We already explained in the conclusion of Chapter 11 that N can be composed of the biggest stakeholders of the network, just like in a PoS system, but this idea needs further research.

A long-term objective for the PoI is to create a blockchain implementation using it. Thus allowing us to make measurements and observe the requirements needed to use this consensus algorithm. If we can combine this consensus algorithm with some storage reduction methods presented in Chapter 2, we may reduce the requirements for being a node and we could consider including devices with less computational power and storage resources than the current blockchain full nodes. Still, these new nodes will need good connectivity to execute the consensus algorithm. In Chapter 4, we explained that, in an IoT architecture, it is realistic to assume that gateways or servers can be blockchain full nodes. Gateways have more resources than constrained devices because they have more responsibilities, but they may not be as powerful as a server. For this reason, by reducing the requirements for being a node, more gateways could join the blockchain network. Thus reducing the delay to share data from IoT devices in the blockchain network.

Appendix

Bibliography

Here are the references in citation order.

[1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. 2008 (cited on pages v, x, 1, 7, 10, 91).

[2] Patrick Th Eugster et al. 'The Many Faces of Publish/Subscribe'. In: *ACM Computing Surveys* 35.2 (2003), pp. 114–131. DOI: 10.1145/857076.857078 (cited on pages vi, 2, 33).

[3] Gowri Sankar Ramachandran et al. 'Trinity: A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence'. In: *ICBC 2019 - IEEE International Conference on Blockchain and Cryptocurrency* (2019), pp. 227–235. DOI: 10.1109/BLOC.2019.8751388 (cited on pages vi, 36, 66, 67).

[4] Gowri Sankar Ramachandran, Kwame-Lante Wright, and Bhaskar Krishnamachari. 'Trinity: A Distributed Publish/Subscribe Broker with Blockchain-based Immutability'. In: (2018), pp. 1–8 (cited on pages vi, 36, 67).

[5] Mehmud Abliz and Taieb Znati. 'A guided tour puzzle for denial of service prevention'. In: *2009 Annual Computer Security Applications Conference*. IEEE. 2009, pp. 279–288 (cited on pages x, 91–93).

[6] Nick Szabo. 'Secure property titles with owner authority'. In: *Online at http://szabo. best. vwh. net/securetitle. html* (1998) (cited on pages 1, 13).

[7] Vitalik Buterin et al. 'Ethereum white paper'. In: *GitHub repository* 1 (2013), pp. 22–23 (cited on pages 1, 13, 77).

[8] 'Trade-offs between Distributed Ledger Technology Characteristics'. In: *ACM Computing Surveys* 53.2 (2020). DOI: 10.1145/3379463 (cited on page 7).

[9] Leslie Lamport, Robert Shostak, and Marshall Pease. 'The Byzantine Generals Problem'. In: *ACM Transactions on Programming Languages and Systems* (1982), pp. 382–401 (cited on page 8).

[10] Danny Dolev and H. Raymond Strong. 'Authenticated algorithms for Byzantine agreement'. In: *SIAM Journal on Computing* 12.4 (1983), pp. 656–666 (cited on page 9).

[11] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 'Consensus in the presence of partial synchrony'. In: *Journal of the ACM (JACM)* 35.2 (1988), pp. 288–323 (cited on page 9).

[12] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 'Impossibility of distributed consensus with one faulty process'. In: *Journal of the ACM (JACM)* 32.2 (1985), pp. 374–382 (cited on page 9).

[13] John R Douceur. 'The sybil attack'. In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260 (cited on pages 9, 10).

[14] Adam Back. 'Hashcash - A Denial of Service Counter-Measure'. In: *Http://Www.Hashcash.Org/Papers/Hashcash.Pdf* August (2002), pp. 1–10 (cited on pages 10, 91).

[15] Rafael Pass, Lior Seeman, and Abhi Shelat. 'Analysis of the blockchain protocol in asynchronous networks'. In: *EUROCRYPT 2017* 10211 LNCS (2017), pp. 643–673. DOI: 10.1007/978-3-319-56614-6_22 (cited on page 11).

[16] Florian Tschorsch and Björn Scheuermann. 'Bitcoin and beyond: A technical survey on decentralized digital currencies'. In: *IEEE Communications Surveys and Tutorials* 18.3 (2016), pp. 2084–2123. DOI: 10.1109/COMST.2016.2535718 (cited on page 12).

[17] Sunny King and Scott Nadal. 'Ppcoin: Peer-to-peer crypto-currency with proof-of-stake'. In: *self-published paper, August* 19 (2012) (cited on page 12).

[18] Peter Gaži, Aggelos Kiayias, and Alexander Russell. 'Stake-bleeding attacks on proof-of-stake blockchains'. In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE. 2018, pp. 85–92 (cited on page 12).

[19] François Bonnet, Quentin Bramas, and Xavier Défago. 'Stateless Distributed Ledgers'. In: *arXiv preprint arXiv:2006.10985* (2020) (cited on page 12).

[20] Wellington Fernandes Silvano and Roderval Marcelino. 'Iota Tangle: A cryptocurrency to communicate Internet-of-Things data'. In: *Future Generation Computer Systems* 112 (2020), pp. 307–319. DOI: https://doi.org/10.1016/j.future.2020.05.047 (cited on page 13).

[21] *PoET 1.0 Specification*. https://sawtooth.hyperledger.org/docs/core/releases/1.2.4/architecture/poet.html (cited on page 15).

[22] Lin Chen et al. 'On Security Analysis of Proof-of-Elapsed-Time (PoET)'. In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by Paul Spirakis and Philippas Tsigas. Springer International Publishing (cited on page 15).

[23] Shikah J Alsunaidi and Fahd A Alhaidari. 'A survey of consensus algorithms for blockchain technology'. In: *2019 International Conference on Computer and Information Sciences (ICCIS)*. IEEE. 2019, pp. 1–6 (cited on page 15).

[24] Asutosh Palai, Meet Vora, and Aashaka Shah. 'Empowering Light Nodes in Blockchains with Block Summarization'. In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018 - Proceedings* 2018-Janua (2018), pp. 1–5. DOI: 10.1109/NTMS.2018.8328735 (cited on page 15).

[25] Emanuel Palm, Olov Schelen, and Ulf Bodin. 'Selective blockchain transaction pruning and state derivability'. In: *Proceedings - 2018 Crypto Valley Conference on Blockchain Technology, CVCBT 2018* (2018), pp. 31–40. DOI: 10.1109/CVCBT.2018.00009 (cited on page 15).

[26] Roman Matzutt et al. 'How to Securely Prune Bitcoin's Blockchain'. In: *IFIP Networking 2020 Conference and Workshops, Networking 2020* (2020), pp. 298–306 (cited on page 15).

[27] Roman Matzutt et al. 'CoinPrune: Shrinking Bitcoin's Blockchain Retrospectively'. In: *IEEE Transactions on Network and Service Management* 18.3 (2021), pp. 3064–3078. DOI: 10.1109/TNSM.2021.3073270 (cited on page 15).

[28] Teasung Kim, Jaewon Noh, and Sunghyun Cho. 'SCC: Storage Compression Consensus for Blockchain in Lightweight IoT Network'. In: *2019 IEEE International Conference on Consumer Electronics, ICCE 2019* (2019). DOI: 10.1109/ICCE.2019.8662032 (cited on page 15).

[29] 'Recycling Smart Contracts: Compression of the Ethereum Blockchain'. In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018 - Proceedings* 2018-Janua (2018), pp. 1–5. DOI: 10.1109/NTMS.2018.8328742 (cited on page 15).

[30] Xiaojiao Chen, Sianjheng Lin, and Nenghai Yu. 'Bitcoin Blockchain Compression Algorithm for Blank Node Synchronization'. In: *2019 11th International Conference on Wireless Communications and Signal Processing, WCSP 2019* (2019). DOI: 10.1109/WCSP.2019.8928104 (cited on page 15).

[31] Qiuhong Zheng et al. 'An Innovative IPFS-Based Storage Model for Blockchain'. In: *Proceedings - 2018 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2018* (2019), pp. 704–708. DOI: 10.1109/WI.2018.000-8 (cited on page 15).

[32] Randhir Kumar and Rakesh Tripathi. 'Implementation of Distributed File Storage and Access Framework using IPFS and Blockchain'. In: *Proceedings of the IEEE International Conference Image Information Processing* 2019-Novem (2019), pp. 246–251. DOI: 10.1109/ICIIP47207.2019.8985677 (cited on page 15).

[33] Jakub Sliwinski and Roger Wattenhofer. 'Asynchronous Proof-of-Stake'. In: () (cited on page 15).

[34] Jakub Sliwinski and Roger Wattenhofer. 'ABC: Proof-of-Stake without Consensus'. In: (2019) (cited on page 15).

[35] Miguel Castro, Barbara Liskov, et al. 'Practical byzantine fault tolerance'. In: *OsDI*. Vol. 99. 1999. 1999, pp. 173–186 (cited on page 16).

[36] Joseph Poon and Vitalik Buterin. 'Plasma: Scalable Autonomous Smart Contracts'. In: *Whitepaper* (2017), pp. 1–47 (cited on pages 17, 46).

[37] Rami Khalil et al. 'Commit-Chains : Secure , Scalable Off-Chain Payments'. In: *Cryptology ePrint Archive* i (2018), p. 642 (cited on page 17).

[38] Joseph Poon and Thaddeus Dryja. 'The bitcoin lightning network'. In: *Scalable o-chain instant payments* (2015) (cited on pages 20, 46, 47).

[39] Maurice Herlihy. 'Atomic cross-chain swaps'. In: *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing* (2018), pp. 245–254. DOI: 10.1145/3212734.3212736 (cited on pages 23, 39, 46, 49, 83).

[40] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 'General state channel networks'. In: *Proceedings of the ACM Conference on Computer and Communications Security* (2018), pp. 949–966. DOI: 10.1145/3243734.3243856 (cited on page 23).

[41] Stefan Dziembowski et al. *Multi-party virtual state channels*. Vol. 11476 LNCS. 2019, pp. 625–656 (cited on page 23).

[42] Fernando E Alvarez, David Argente, and Diana Van Patten. *Are Cryptocurrencies Currencies? Bitcoin as Legal Tender in El Salvador*. Tech. rep. National Bureau of Economic Research, 2022 (cited on page 24).

[43] Gabriel Montenegro et al. 'Transmission of IPv6 packets over IEEE 802.15. 4 networks'. In: *Internet proposed standard RFC* 4944 (2007), p. 130 (cited on page 25).

[44] Carsten Bormann, Mehmet Ersue, and Ari Keranen. 'Terminology for constrained-node networks'. In: *Internet Engineering Task Force (IETF): Fremont, CA, USA* (2014), pp. 2070–1721 (cited on page 25).

[45] Bitcoin Project. *Bitcoin Core requirements*. https://bitcoin.org/en/bitcoin-core/features/requirements. 2022 (cited on page 25).

[46] Ethereum Foundation. *Geth requirements*. https://ethereum.org/en/developers/docs/nodes-and-clients/. 2022 (cited on page 25).

[47] IOTA Foundation. *Hornet requirements*. https://wiki.iota.org/hornet/getting_started. 2022 (cited on page 25).

[48] Yash Gupta et al. 'The applicability of blockchain in the Internet of Things'. In: *2018 10th International Conference on Communication Systems and Networks, COMSNETS 2018* 2018-Janua (2018), pp. 561–564. DOI: 10.1109/COMSNETS.2018.8328273 (cited on page 26).

[49] Madhusudan Singh, Abhiraj Singh, and Kim Shiho. 'Blockchain: A game changer for securing IoT data'. In: *IEEE World Forum on Internet of Things, WF-IoT 2018 - Proceedings* 2018-Janua (2018), pp. 51–55. DOI: 10.1109/WF-IoT.2018.8355182 (cited on page 26).

[50] Otto Julio Ahlert Pinno, Andre Ricardo Abed Gregio, and Luis C.E. De Bona. 'ControlChain: Blockchain as a Central Enabler for Access Control Authorizations in the IoT'. In: *2017 IEEE Global Communications Conference, GLOBECOM 2017 - Proceedings* 2018-Janua (2018), pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254521 (cited on page 26).

[51] M. Shyamala Devi, R. Suguna, and P. M. Abhinaya. 'Integration of Blockchain and IoT in Satellite Monitoring Process'. In: *Proceedings of 2019 3rd IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2019* (2019), pp. 1–6. DOI: 10.1109/ICECCT.2019.8869185 (cited on page 26).

[52] Pradip Kumar Sharma, Seo Yeon Moon, and Jong Hyuk Park. 'Block-VN: A distributed blockchain based vehicular network architecture in smart city'. In: *Journal of Information Processing Systems* 13.1 (2017), pp. 184–195. DOI: 10.3745/JIPS.03.0065 (cited on page 26).

[53] Junqin Huang et al. 'B-IoT: Blockchain driven internet of things with credit-based consensus mechanism'. In: *Proceedings - International Conference on Distributed Computing Systems* 2019-July (2019), pp. 1348–1357. DOI: 10.1109/ICDCS.2019.00135 (cited on page 27).

[54] Jun Lin et al. 'Using blockchain to build trusted LoRaWAN sharing server'. In: *International Journal of Crowd Science* 1.3 (2017), pp. 270–280. DOI: 10.1108/ijcs-08-2017-0010 (cited on page 27).

[55] Sina Rafati Niya et al. 'Adaptation of Proof-of-Stake-based Blockchains for IoT Data Streams'. In: c (2019), pp. 15–16. DOI: 10.1109/bloc.2019.8751260 (cited on page 27).

[56] Bin Liu et al. 'Blockchain Based Data Integrity Service Framework for IoT Data'. In: *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017* (2017), pp. 468–475. DOI: 10.1109/ICWS.2017.54 (cited on pages 27, 30).

[57] Ali Dorri et al. 'Blockchain for IoT security and privacy: The case study of a smart home'. In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2017* (2017), pp. 618–623. DOI: 10.1109/PERCOMW.2017.7917634 (cited on page 27).

[58] Seyoung Huh, Sangrae Cho, and Soohyung Kim. 'Managing IoT devices using blockchain platform'. In: *International Conference on Advanced Communication Technology, ICACT* (2017), pp. 464–467. DOI: 10.23919/ICACT.2017.7890132 (cited on pages 27, 28).

[59] Oscar Novo. 'Scalable access management in IoT using blockchain: A performance evaluation'. In: *IEEE Internet of Things Journal* 6.3 (2019), pp. 4694–4701. DOI: 10.1109/JIOT.2018.2879679 (cited on pages 28, 29).

[60] Oscar Novo. 'Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT'. In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 1184–1195. DOI: 10.1109/JIOT.2018.2812239 (cited on page 28).

[61] Yuanyu Zhang et al. 'Smart contract-based access control for the internet of things'. In: *IEEE Internet of Things Journal* 6.2 (2019), pp. 1594–1605. DOI: 10.1109/JIOT.2018.2847705 (cited on pages 28, 29).

[62] 'IoTChain: A blockchain security architecture for the Internet of Things'. In: *IEEE Wireless Communications and Networking Conference, WCNC* 2018-April (2018), pp. 1–6 (cited on page 28).

[63] Mathis Steichen et al. 'Blockchain-Based, Decentralized Access Control for IPFS'. In: *Proceedings - IEEE 2018 International Congress on Cybermatics: 2018 IEEE Conferences on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, iThings/Gree* (2018), pp. 1499–1506. DOI: 10.1109/Cybermatics_2018.2018.00253 (cited on page 28).

[64] Huan Zhou et al. 'A Blockchain based Witness Model for Trustworthy Cloud Service Level Agreement Enforcement'. In: *Proceedings - IEEE INFOCOM* 2019-April (2019), pp. 1567–1575. DOI: 10.1109/INFOCOM.2019.8737580 (cited on pages 30, 107).

[65] Yunlei Sun et al. 'A low-delay, lightweight publish/subscribe architecture for delay-sensitive IOT services'. In: *Proceedings - IEEE 20th International Conference on Web Services, ICWS 2013* (2013), pp. 179–186 (cited on page 33).

[66] Michele Amoretti et al. 'A scalable and secure publish/subscribe-based framework for industrial IoT'. In: *IEEE Transactions on Industrial Informatics* 17.6 (2020), pp. 3815–3825 (cited on page 33).

[67] Aleksandar Antonić et al. 'A mobile crowd sensing ecosystem enabled by CUPUS: Cloud-based publish/subscribe middleware for the Internet of Things'. In: *Future Generation Computer Systems* 56 (2016), pp. 607–622 (cited on page 34).

[68] 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. Accessed: 2020-06-09. 2019 (cited on page 34).

[69] Markus Dahlmanns et al. *Transparent End-to-End Security for Publish/Subscribe Communication in Cyber-Physical Systems*. Vol. 1. 1. Association for Computing Machinery, 2021, pp. 78–87 (cited on page 34).

[70] Sam Kumar et al. 'Jedi: Many-to-many end-to-end encryption and key delegation for IoT'. In: *Proceedings of the 28th USENIX Security Symposium* (2019), pp. 1519–1536 (cited on page 34).

[71] Cristian Borcea et al. 'PICADOR: End-to-end encrypted Publish–Subscribe information distribution with proxy re-encryption'. In: *Future Generation Computer Systems* 71 (2017), pp. 177–191 (cited on page 34).

[72] Shrideep Pallickara et al. 'A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems'. In: *2006 7th IEEE/ACM International Conference on Grid Computing*. 2006, pp. 215–222. DOI: 10.1109/ICGRID.2006.311018 (cited on page 34).

[73] Fengyun Cao and Jaswinder Pal Singh. 'Efficient event routing in content-based publish-subscribe service networks'. In: *Proceedings - IEEE INFOCOM* 2.C (2004), pp. 929–940. DOI: 10.1109/infcom.2004.1356980 (cited on page 35).

[74] Silvia Bianchi, Pascal Felber, and Maria Gradinariu Potop-Butucaru. 'Stabilizing distributed R-trees for peer-to-peer content routing'. In: *IEEE Transactions on Parallel and Distributed Systems* 21.8 (2010), pp. 1175–1187. DOI: 10.1109/TPDS.2009.131 (cited on page 35).

[75] Rahul Shah et al. 'Efficient dissemination of personalized information using content-based multicast'. In: *IEEE Transactions on Mobile Computing* 3.4 (2004), pp. 394–408 (cited on page 35).

[76] Spyros Voulgaris, Etienne Rivière, and Anne-marie Kermarrec Maarten. 'S UB -2-S UB : Self-Organizing Content-Based Publish and Subscribe for Dynamic and Large Scale'. In: *IPTPS'06: the fifth International Workshop on Peer-to-Peer Systems* (2005), p. 16 (cited on page 35).

[77] João Paulo De Araujo et al. 'A Publish/Subscribe System Using Causal Broadcast over Dynamically Built Spanning Trees'. In: *Proceedings - 29th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2017* (2017), pp. 161–168. DOI: 10.1109/SBAC-PAD.2017.28 (cited on page 35).

[78] Shrideep Pallickara et al. 'A security framework for distributed brokering systems'. In: *Community Grids Laboratory Technical Report. Available from http://www. naradabrokering. org* (2003) (cited on pages 35, 36).

[79] Mudhakar Srivatsa, Ling Liu, and Arun Iyengar. 'Eventguard: A system architecture for securing publish-subscribe networks'. In: *ACM Transactions on Computer Systems (TOCS)* 29.4 (2011), pp. 1–40 (cited on page 35).

[80] Gianluca Dini and Angelica Lo Duca. 'On securing publish-subscribe systems with security groups'. In: *2009 IEEE Symposium on Computers and Communications*. IEEE. 2009, pp. 532–537 (cited on page 35).

[81] Pin Lv et al. 'An IOT-oriented privacy-preserving publish/subscribe model over blockchains'. In: *IEEE Access* 7 (2019), pp. 41309–41314. DOI: 10.1109/ACCESS.2019.2907599 (cited on pages 36, 37, 67).

[82] Gewu Bu et al. 'HyperPubSub: Blockchain based publish/subscribe'. In: *Proceedings of the IEEE Symposium on Reliable Distributed Systems* (2019), pp. 366–368. DOI: 10.1109/SRDS47363.2019.00052 (cited on pages 37, 41, 77).

[83] Elli Androulaki et al. 'Hyperledger fabric: a distributed operating system for permissioned blockchains'. In: *Proceedings of the thirteenth EuroSys conference*. 2018, pp. 1–15 (cited on pages 37, 41).

[84] R. Radhakrishnan and B. Krishnamachari. 'Streaming Data Payment Protocol (SDPP) for the Internet of Things'. In: (2018), pp. 1679–1684. DOI: 10.1109/Cybermatics_2018.2018.00280 (cited on pages 39, 48, 86).

[85] David Chen et al. 'PayFlow: Micropayments for bandwidth reservations in software defined networks'. In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2019, pp. 26–31 (cited on page 40).

[86] Ryota Nakada, Kien Nguyen, and Hiroo Sekiya. 'Implementation of Micropayment System Using IoT Devices'. In: *Journal of Signal Processing* 25.4 (2021), pp. 137–140 (cited on page 40).

[87] Gowri Sankar Ramachandran et al. 'Publish-pay-subscribe protocol for payment-driven edge computing'. In: *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*. 2019 (cited on pages 41, 77).

[88] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. 'MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks'. In: *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. IEEE. 2008, pp. 791–798 (cited on page 58).

[89] *Transmission Control Protocol*. RFC 793. 1981 (cited on page 66).

[90] Anton V. Uzunov. 'A survey of security solutions for distributed publish/subscribe systems'. In: *Computers and Security* 61 (2016), pp. 94–129 (cited on page 66).

[91] Quentin Bramas. *Proof-of-Concept of SUPRA, a distributed publish/subscribe protocol with blockchain as a conflict resolver*. https://github.com/Bramas/supra-PoC. 2022 (cited on page 67).

[92] Syed Muhammad Danish et al. 'A Lightweight Blockchain Based Two Factor Authentication Mechanism for LoRaWAN Join Procedure'. In: *2019 IEEE International Conference on Communications Workshops (ICC Workshops)* (2019), pp. 1–6. DOI: 10.1109/iccw.2019.8756673 (cited on page 67).

[93] Ji-Sun Park et al. 'Smart contract-based review system for an IoT data marketplace'. In: *Sensors* 18.10 (2018), p. 3577 (cited on page 85).

[94] Wenbo Wang et al. 'A survey on consensus mechanisms and mining strategy management in blockchain networks'. In: *IEEE Access* 7 (2019), pp. 22328–22370 (cited on page 91).

[95] Bart Preneel. 'Analysis and design of cryptographic hash functions'. PhD thesis. Katholieke Universiteit te Leuven, 1993 (cited on page 92).

[96] Suratose Tritilanunt et al. 'Toward Non-parallelizable Client Puzzles'. In: *Cryptology and Network Security*. Ed. by Feng Bao et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 247–264 (cited on page 96).

[97] Ittay Eyal and Emin Gün Sirer. 'Majority is not enough: Bitcoin mining is vulnerable'. In: *International conference on financial cryptography and data security*. Springer. 2014, pp. 436–454 (cited on page 101).

[98] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. 'Optimal selfish mining strategies in bitcoin'. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 515–532 (cited on page 101).

[99] Jae Kwon. 'Tendermint: Consensus without mining'. In: *Draft v. 0.6, fall* 1.11 (2014) (cited on page 102).

[100] Miguel Castro and Barbara Liskov. 'Practical Byzantine fault tolerance and proactive recovery'. In: *ACM Transactions on Computer Systems (TOCS)* 20.4 (2002), pp. 398–461 (cited on page 103).

[101] EOS IO. 'Eos. io technical white paper'. In: *EOS. IO (accessed 18 December 2017) https://github. com/EOSIO/Documentation* (2017) (cited on page 103).

[102] Erik Daniel and Florian Tschorsch. 'IPFS and friends: A qualitative comparison of next generation peer-to-peer data networks'. In: *IEEE Communications Surveys & Tutorials* 24.1 (2022), pp. 31–52 (cited on page 107).

Special Terms

D

DAG Directed Acyclic Graph. 26

DLT Distributed Ledger Technologies. 7, 8, 25, 26, 28, 31, 34–36, 42, 91

H

HTLC Hashed Timelock Contract. 21–23

I

IoT Internet of Things. xix, 1, 2, 24–28, 31, 33, 37, 39, 77, 86, 87, 105–108

L

LR-WPAN Low Rate Wireless Personal Area Network. 25

P

PKI Public Key Infrastructure. 9

PoI Proof of Interaction. 2, 3, 105–108

PoS Proof of Stake. 12, 14, 108

PoW Proof of Work. 2, 3, 10–12, 14, 15, 25, 31, 46, 91, 106

S

SUPRA Secured Update Protocol with Righterous Accusations. xix, 2, 45–48, 51, 53, 55, 56, 61, 64–69, 76, 77, 86, 87, 105–107

Abstracts

Résumé

Cette thèse traite de l'utilisation de la Blockchain dans les applications liées à l'Internet des Objets (IoT). L'IoT est un domaine présent dans notre quotidien et les applications IoT reposent sur l'utilisation d'objets contraints, des appareils informatiques beaucoup moins puissants qu'un ordinateur. L'utilisation de ces objets contraints force la définition de nouveaux protocoles et l'utilisation de nouveaux outils pour assurer les communications et la sécurité des applications. Dans cette optique, nous nous posons la question de savoir si la blockchain, une technologie de registre distribuée, peut être un outil permettant de répondre à ce problème de sécurité.

Pour répondre à cette question, nous avons réalisé deux contributions: un algorithme de consensus réduisant la consommation énergétique des nœuds blockchain, et un protocole publish/subscribe fournissant des garanties de livraison des données.

Mots clés : blockchain, internet des objets, publish/subscribe, algorithme de consensus, paiement des données

Abstract

This thesis presents how the blockchain technology is used in Internet-of-Things (IoT) applications. IoT is something present in our everyday life and the IoT applications rely on constrained devices, devices far less powerful than a computer. Using these constrained devices forces us to define new protocols and use new tools to ensure security in these applications. In this thesis, we ask ourselves if the blockchain, a distributed ledger technology, can be a tool used to increase security in IoT applications.

To answer this question, we propose two contributions : a consensus algorithm reducing the energy consumption of the blockchain nodes, and a publish/subscribe protocol presenting data delivery guarantees.

Keywords : blockchain, internet-of-things, publish/subscribe, consensus algorithm, data payment protocol