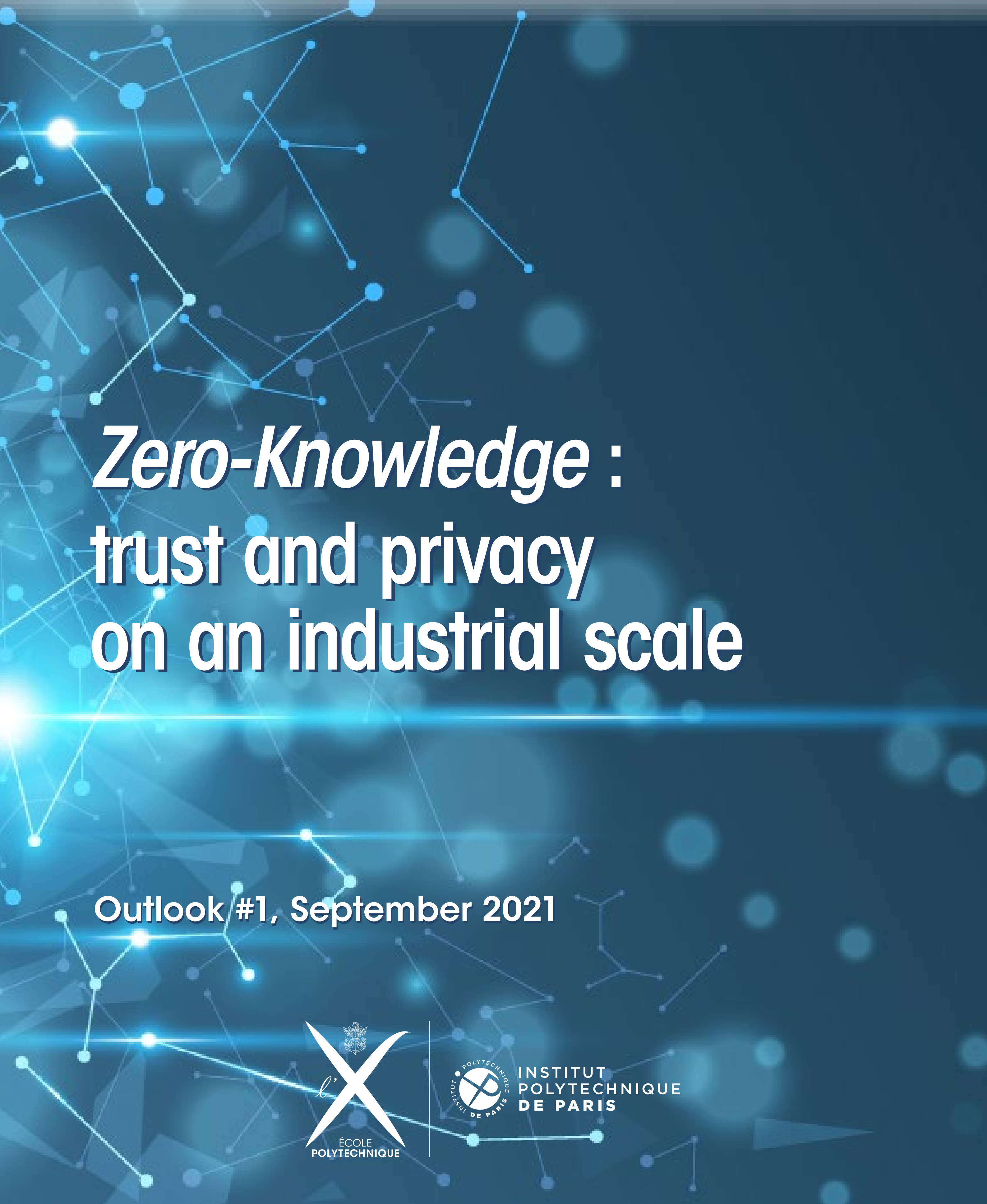
Zero-Knowledge : trust and privacy on an industrial scale

Daniel Augot, Sarah Bordage, Youssef El Housni, Gilles Fedak, Anthony Simonet

BLOCKCHAIN& PLATFORM CHAIR



Zero-Knowledge : trust and privacy on an industrial scale

Outlook #1, September 2021

l'X ÉCOLE POLYTECHNIQUE

INSTITUT POLYTECHNIQUE DE PARIS

Introduction

One of the main obstacles to the deployment of blockchains is the fact that the data managed on a blockchain is publicly accessible. This is unthinkable in the health or banking sectors, for example.

Zero-knowledge technologies can resolve precisely this contradiction. These technologies can be implemented either by deploying a blockchain specifically designed to integrate zero-knowledge, or by deploying zero-knowledge software components on an existing blockchain that is technologically capable of integrating them.

This concept of zero-knowledge is so amazing and promising that it earned its inventors, Shafi Goldwasser and Silvio Micali, the Turing Award in 2012.

To present this counter-intuitive technology and put it into perspective with uses and services, the “**Blockchain and B2B Platforms**” chair hosted at **École Polytechnique** and supported by **CapGemini**, **NomadicLabs** and **Caisse des Dépôts**, chose to interview two doctoral students, Sarah Bordage and Youssef El Housni, who are doing their research at the **École Polytechnique**’s computer science laboratory as part of the chair, and Anthony Simonet-Boulogne and Gilles Fedak, from the startup **iExec**, who see zero-knowledge as technology to be integrated into their offer.

What is "zero-knowledge"?

Daniel Augot : In the world of blockchains and cryptocurrencies, we hear a lot about zero-knowledge as a solution to privacy issues, i.e. confidentiality and privacy. Indeed, this technology can satisfy the two conflicting goals of make information public (or recording or sharing information) and, on the other hand, keeping that information confidential.

In a very simplified way, the principle is as follows: a minimal "trace" or "fingerprint" (cryptographic **commitment**[1], or pledge) of a piece of information is recorded on a blockchain, whether public or not (for example a transaction). On the basis of this trace it is possible to prove facts about this information without revealing the information itself. A pledge, or **commitment**, of an item of information consists of producing a seal also called a **hash**, possibly a very short one, which means that only this information is certified by this **hash**. This **hash** can also hide the certified information.

The applications are: confidentiality of financial transactions on a blockchain, identity management, proof of solvency, protection of health data, etc.

How is this possible? It seems counter-intuitive... A playful presentation based on Waldo's game is presented in the box.

Where is Waldo ?

The game is well known: it is one of those books with large, dense and detailed illustrations. Waldo hides in them, easily recognisable, once he has been found. A child (the prover) will prove to his father (the verifier) that he has found Waldo without showing where Waldo is.

He proceeds as follows:

1. he prepares a card as big as nine times the page
2. he cuts out the silhouette of Waldo from this card
3. he presents this card to his father
4. he positions the book behind the card so that Waldo appears in the cut-out
5. the father sees Waldo through the cut-out but does not see Waldo's position on the page.

The father is convinced that his child knows where Waldo is, without learning where Waldo is.

[1] Words in bold font are explained in text or in the glossary

What does zero-knowledge bring to data privacy?

Daniel Augot : Generally, when we speak of data confidentiality, we are referring to **encryption**, i.e. a cryptographic algorithm that makes the data unintelligible, using an **encryption key**. The only way to retrieve the data is to decipher it, using the **decryption key**. Thus, the legitimate recipient can decipher the data with the **decryption key**. It's an all-or-nothing solution, either you know nothing or you see everything, and possession of the **decryption key** determines who has access to plaintext information.

In contrast, zero-knowledge proofs can be used to prove the truth of **statements** about data that have been kept hidden by encryption (or **commitment**, cryptographic **hashing** or pledging). These proofs do not reveal any information other than the fact that these properties or **statements** are true. For example, if a digital fingerprint of passport data is stored, the user of that passport will be able to prove that they are of age without revealing their age or other passport information. In the banking world, bank accounts would be recorded and accompanied by the publication of a public fingerprint (the root of a **Merkle tree**. A **Merkle tree** allows a large amount of data to be pledged with a very short fingerprint.) Based on this fingerprint, the bank can prove that all accounts do not exceed a certain reporting threshold without revealing the status of the accounts.

Recent advances have made it possible to move beyond original zero-knowledge, which historically dealt with abstract, mathematical **statements** from number theory. Now, one can deal with real-life **statements**, such as those described above for age or for the threshold for reporting bank accounts. Essentially, it is a kind of compilation of a high-level language into mathematical equations (see box).

A second important advance, thanks to this increased flexibility, is to allow proofs that calculations have been carried out correctly, without having to redo the calculation, and without revealing all the information necessary for this calculation. These calculations involve data, some public, some private, and it can be proven that the result is correct without revealing the private data. To go beyond the previous banking example of verifying a static threshold, the bank has the possibility of producing a more dynamic calculation result, such as the median or average of the accounts under its management, accompanied by a zero-knowledge proof that this calculation is correct. In short, someone who has made a complex calculation on data can convince that this result is correct by producing the associated proof, but without revealing the data in question.

Technical aspects of zero knowledge proof

First a bit of terminology: the prover is the person who produces a convincing proof (in Waldo's example this is the child), and the verifier is the person who checks the proof (in Waldo's example this is the parent). The prover uses a proof algorithm to construct the proof and the verifier uses a verification algorithm to verify the proof.

Historically, since the 1980s, cryptographers have built low-level zero-knowledge protocols, which make it possible, among other things, to prove that a system of algebraic equations has a solution, that a «discrete logarithm» is known, etc. These low-level protocols are expressed in mathematical and computational terms, far removed from the concrete problems of everyday life.

But we should remember that in computing, all information processing comes down to manipulating bits, i.e. 0s and 1s. To make the programmer's job easier, a computer program is written in a high-level language, such as Rust or C++, and then compiled into a low-level language that the machine can understand.

Here, the situation is the same: a natural **statement** requiring a natural proof is compiled into a mathematical **statement**. The natural proof is likewise compiled into a low-level mathematical proof, which will be processed by the low-level cryptographic and mathematical zero-knowledge protocol. Verifying the proof of a high-level **statement** in zero-knowledge is the same as verifying the associated low-level proof in zero-knowledge.

It is said that the mathematical problem is the **backend** and the high-level language is the **frontend**. However, the lower cryptographic layer may have specifics depending on the proof system that impact higher level applications. In this talk, we consider SNARKs (Succinct Non Interactive Argument of Knowledge) and STARKs (Scalable Transparent ARguments of Knowledge) which are the two most industrially advanced technologies.

It is a scientific, technological and engineering feat to integrate cryptographic protocols into a software suite making it easy for the programmer to express natural problems. Moreover, there is an effort to standardise these systems and protocols, bringing industrialists and academics together.

What are the criteria for choosing zero-knowledge systems for different applications?

Youssef El Housni : For a given application, this question can be dealt with according to two axes of analysis, depending on the technological constraints of the environment in which the application will be deployed or according to the functionalities and needs of the application.

On the first axis, to analyse according to technological constraints, the question in the context of blockchains and **smart contracts** is to know how and in what environment the proof verification algorithm will be executed. A **smart contract** is a program that is irrevocably stored in the blockchain under consideration, and whose execution is triggered automatically. It typically enables more complex transactions than simple financial transactions, and the development of more sophisticated financial tools.

For example, for Ethereum, the Ethereum virtual machine that executes the **smart contracts** must be pre-compiled with non-standard complex instructions. These non-standard complex instructions implement, among other things, advanced elliptic curve operations needed to verify a zero-knowledge proof on-chain. They specify a rigid mathematical context, such as a specific well-defined elliptic curve, which then restricts the whole proof system and sets a technological limit. Indeed, changing the curve would imply a heavy change consisting in adding new precompiled instructions to the Ethereum virtual machine.

On the other hand, for a new application or a new blockchain with no prior context, the freedom of choice of the elliptic curve makes it possible to incorporate the most recent advances in research.

Sarah Bordage : For **STARKs**, the platform requirements are relatively light, requiring a library of algorithms for addition and multiplication of large numbers and polynomials. This is not as specific and mathematically complex as choosing an elliptic curve for **SNARKs**.

Youssef El Housni : For the second axis, to consider a zero-knowledge system according to the functionalities of the application, it is a question of characterising the proofs in relation to their size, the time taken to generate the proofs and the time taken to verify the proofs. In an application where evidence is stored permanently after verification, the size of the proof is of primary importance. It is therefore necessary that these proofs are as small as possible, such as those of **SNARKs**. If the application requires the prover to make proofs quickly, **STARKs** with a faster proof algorithm than **SNARKs** should be used.

On the other hand, a zero-knowledge algorithm, whose verification times depend on the **statement** and are not constant, would pose problems for use in an application that verified different **statements** of varying sizes. This would greatly impact the performance of the application, which would no longer be in real time. **SNARKs** have constant verification times, so have less impact, on the verifier's side, on the performance of the applications that use them.

For example, in **Zcash**, proofs of each transaction are stored on the blockchain and are verified by miners: to keep them short and quickly verifiable, **SNARKs** are used to control the rate of block production.

	SNARKs	STARKs
Trusted Setup	YES	NO
Universal Trusted Setup	YES	Non applicable
Proof Calculation Cost	Fast	Very fast
Proof Checking Cost	Constant	Logarithmic
Proof size	Constant	Logarithmic

Anthony Simonet-Boulogne: From my point of view, these different zero-knowledge protocols enable us to adjust the cursor between the traceability of transactions on the one hand, which is essential in blockchain applications, and the confidentiality of data on the other. In the end, it is therefore the specific needs of each application that will guide our choice between the various solutions.

For example, the time taken to generate proofs can be very restrictive in certain situations. A payment system such as **Zcash** needs extremely fast proofs to be generated as it needs to be able to issue transactions almost instantly, possibly on a light terminal (smartphone). On the other hand, I think there are many application areas, such as **oracles** (an actor entering external information into the blockchain, which intrinsically is not verifiable by miners or validators with only the information in the blockchain, e.g. a local weather forecast), where the trust in the external information that is inserted into the blockchain is so great, that one could accept much longer proof generation times to certify this external information introduced by an **oracle**.

Youssef El Housni: It is very important to consider the issue of trust in the implementation of SNARKs. Indeed, the person who generates the **structured reference string**, the **proof** and **verification algorithms** manipulates secrets that allow him to make false proofs. He must therefore be trusted to have destroyed these secrets. In an application where there is only one type of **statement** to prove, a **SNARK** is relevant because its **trusted setup** only needs to be done once. Thus for the Zcash cryptocurrency, where transactions are secret, there is only one type of proof that is used to verify the validity of transactions according to the criteria of the **Zcash** protocol: no money creation, no double spending, legitimacy of the issuer of the transaction, etc. Once these criteria have been specified, and thus the type of **statement** to be verified (we are referring to the **language** of valid transactions), everything is set in stone and only one implementation is needed for the associated proof system. The **SNARK** solution is appropriate.

On the other hand, in a **smart contract** application on Ethereum, which verifies a zero-knowledge proof on-chain, depending on the logic of the **smart contract**, it would be necessary to set up a **structured reference string** for each new **smart contract** that has to verify zero-knowledge proofs. The variability of the **statements** to be verified and the complexity of their implementation make it unreasonable to deploy such a **trusted setup** for each **smart contract**. It would then be appropriate to consider **STARKs**, which do not require such a set-up of the initial trust framework.

SNARKs : Succinct Non Interactive Arguments of Knowledge

Youssef El Housni: My thesis topic is **SNARKs**. The implementation of **SNARKs** requires a trusted third party. This trusted third party generates data called the **structured reference string**, which is a string of bytes needed for the proof and verification algorithms to work. This **structured reference string** is said to be **trapped**, which means that secrets (discrete logarithms) were needed to set up this string, which is however public. Whoever knows these secrets has the means to fabricate false evidence: a **trapdoor**. He must therefore be trusted not to use this power and to have destroyed the secrets once the **structured reference string** has been published. This **trapdoor** is colloquially called **toxic waste**. Moreover, the result of the implementation is specific to the proven **statement**: it must be redone for each type of **statement** (**language**). However, cryptographic advances have made it possible to construct universal **SNARKs**, with a valid implementation for several **statements**, e.g. **PLONK**.

In mathematical terms, **SNARKs** are based on number theory : elliptic curves over finite fields (these objects allow rudimentary arithmetic, limited to addition), pairings (allow more advanced operations, such as multiplication), as well as the associated algorithmic and cryptographic problems. A natural problem can thus be coded into an additions and multiplications problem involving hidden objects. The **structured reference string** is actually a set of elliptic curve points, whose discrete logarithm are known by the trusted third party building the **structured reference string**.

Once set up for a given **statement**, any prover can use the proof algorithm and associated **structured reference string** to produce proof of a statement without revealing what makes the **statement** true. Any verifier can use the verification algorithm, the proof and the **structured reference string** to verify that the **statement** is true.

The size of the **SNARK** proof is very small (a few hundred bytes) and does not depend on the complexity of the **statement** to be proved (this complexity is however found in the **structured reference string** which grows with the **statement** to be proved). Consequently, the **SNARK** verification algorithm is not only very fast but also has a constant cost with respect to the size of the **statement**.

SNARKs have been made popular in the blockchain world by the advent of the cryptocurrency **Zcash**. In the **Zcash** protocol, users do not broadcast their monetary transactions in plaintext but broadcast an opaque version of them, of which only a **commitment** will be made public, associated with a zero-knowledge proof. This zero-knowledge proof allows miners to publicly verify that the transaction is correct, even though the sender, receiver and amounts are hidden.

The small size of **SNARK** proofs and their high speed of verification mean that they can be verified and written to the **Zcash** blockchain without impacting the performance of the blockchain miners. The issue of the trusted third party when setting up the **structured reference string** presents a crucial problem. Indeed, the person or persons who know the **trapdoor** can make invalid transactions whose proof will nevertheless be accepted as correct. In the **Zcash** framework, it is always the same type of **statement** that is proven, the one defined by the validity restrictions of the transactions. This problem only arises once here.

To remedy this, developers have resorted to complex distributed protocols known as MPC (**secure multiparty computation**). An MPC protocol allows multiple participants to collaborate on a common calculation, depending on each participant's secret, without the participants revealing their secrets to each other.

Each participant contributes to the protocol with their own secret, which results in the establishment of the **structured reference string** without any of them knowing the global secret (the **trapdoor**) associated with this **structured reference string**.

Consensys, my employer, is developing the «gnark» library, which autonomously allows the implementation of the **structured reference string**, the proof algorithm, the verification algorithm, for **statements** written in the Go language. Higher application layers specific to blockchains can use the gnark library.

Why is zero-knowledge often mentioned as a solution to the problem of scalability?

Youssef El Housni: We talk about zero-knowledge as a solution to ensure data confidentiality but it can also be used to solve the problem of scalability. Previously, we talked about the types and families of zero-knowledge that can be characterised by the size of the proof. **Rollups** are a direct application of **SNARKs** (or **STARKs**) thanks to their very short proofs (constant for **SNARKs**, logarithmic in the size of the computation for **STARKs**).

The principle of **rollup**, as the name suggests, is to group together numerous off-chain transactions that will be recorded in a single short transaction on the blockchain, accompanied by the proof that all these transactions are correct. This solves the scalability problem by increasing bandwidth: a single short on-chain transaction actually encodes thousands of transactions, with a tiny size of the proof of correctness of these transactions. This takes up little of the blockchain's space and miners can quickly ensure that all these transactions are correct.

So it is not primarily the zero-knowledge component that is important here. It is mainly the fact that the zero-knowledge proof is of constant or logarithmic size that is interesting.

STARKs : scalable transparent arguments of knowledge

Sarah Bordage: Like **SNARKs**, the **STARKs** that constitute my thesis topic are proofs that accompany the result of a computation to certify that the computation has been correctly performed. **STARKs** can be zero-knowledge, in which case the verifier gains no information about a secret input to the computation. These are known as **ZK-STARKs**.

Beyond the issues of confidentiality and privacy, **STARK** proofs are positioned as a solution to the scalability problem that naturally emerges when moving from a centralised network to a peer-to-peer network, such as a blockchain. The cost of verifying a **STARK** proof is logarithmic in the size of the computation, and a prover can handle a large batch of transactions, produce a single **STARK** proof certifying the validity of all transactions. This proof can then be verified on-chain by miners using very little computing power. In a demonstration on the Ethereum network, the StarkEx system generated a single **STARK** proof to verify 300,000 transactions, at a rate of 3,000 transactions per second, with a cost of 315 gas per transaction.

ZK-STARKs have several advantages over **ZK-SNARKs**. Firstly, their security does not depend on trusting the correct execution of a **trusted setup** algorithm: there is no **structured reference string**, nor a secret **trapdoor** that could be exploited to forge valid proofs of false claims.

In terms of comparison, **SNARKs** provide proofs of constant size, shorter than **STARKs**, due to the set-up phase of the **structured reference string** that encodes the full complexity of the associated **statement**. This setup phase, performed during the execution of the **trusted setup**, is specific to the program to be verified. The counterpart of **STARKs** is that the proofs are longer than those of **SNARKs**. However, they are extremely short: logarithmic in the size of the computation to be proven.

If the security of the zero-knowledge application has to withstand the quantum computer, for example for very long-term security, one should turn to **STARKs** for which there is probably no quantum attack.

STARKs provide a great deal of flexibility, especially with the Cairo development platform, recently developed by the start-up Starkware. Cairo is the first language that allows writing generic programs that can be verified by **STARK** proofs. Executing a program written in Cairo returns an execution trace that is sent to a **STARK** prover. From this trace, the proof algorithm generates a **STARK** proof to certify the validity of the computation represented by the Cairo program. For blockchain applications, the proof algorithm is typically executed outside the blockchain network (off-chain). In contrast, a **STARK** verification algorithm can be deployed, without **trusted setup**, via a single **smart contract** for any proof, capable of validating the execution of any Cairo program.

STARKs are used in production in the StarkEx system: a set of **smart contracts** deployed on Ethereum. It is a **rollup** that allows for the on-chain verification of a whole batch of off-chain transactions, of which a **STARK** proof is provided by the system. Decentralised finance applications (DeFi) such as dYdX (decentralised exchanges), DeversiFi (trades and swaps) and Immutable (blockchain gaming) in turn use the Starkex platform.

How does the price of Ethereum gas impact the development and use of zero-knowledge?

Youssef El Housni : Remember that in the case of **SNARKs**, there are three algorithms: a setup algorithm, a proof generation algorithm and a proof verification algorithm. The first two algorithms for setting up and generating the proof are not done on the blockchain (they use secret quantities and data). Only the proof verification algorithm is executed on the blockchain by the miners or verifiers. It is therefore this last algorithm that will impact the cost of using zero-knowledge on a blockchain. In the case of a blockchain, such as Ethereum, where you have to pay for gas to verify a proof, we would like the verification to be as cheap as possible in gas. Ideally, verification should always have the same cost for any type of **statement** to be verified, regardless of its complexity. This is the case with **SNARKs**, which allow us to have a constant cost of proof verification in terms of operations, irrespective of the **statement** being proven, and therefore a perfectly controlled on-chain cost of proof verification.

Sarah Bordage : In contrast, in the case of **STARKs**, the cost of verification, although logarithmic, depends on the complexity of the **statement** to be verified. In the case of Ethereum, the on-chain cost varies according to the **statement** (the logic of the **smart contract**) and is no longer constant, albeit very small. This is the counterpart of the transparency provided by **STARKs**, which do not require a **trusted setup** and allow deployment without the need for trust.

Zcash cryptocurrency: a first example of large-scale deployment

The **Zcash** cryptocurrency, based on the **zerocash** protocol, has helped to introduce the term **ZK-SNARK** and the notion of zero-knowledge to a large audience. This cryptocurrency allows for transparent transactions like Bitcoin or Ethereum or opaque transactions. For opaque transactions, Zcash is based on the notion of a note, hidden by its commitment. A **Zcash** note describes in a non-explicit way an amount and a payment address, in and out. A **private key** is assigned to each payment address, allowing the note to be spent. This information is hidden: the amount and the address of the recipient of the payment are known only to the person who creates the note.

Each note has a **commitment** associated with it, and a **nullifier** that can only be computed with the **private key** associated with the note. It is impossible to link a **nullifier** to a **commitment** without knowing the corresponding **private key**.

A valid consumable note at a given time is a note whose **commitment** exists publicly and whose **nullifier** does not. The system publicly records **commitments** and **nullifiers**. The system is considered here as orthogonal to this zero-knowledge payment system. Its role is to check the validity of the transactions, which handle the notes, as shown below.

A transaction covertly describes an incoming A note and an outgoing B note. Specifically, it reveals the N_A nullifier of the spent note, and reveals a C_B **commitment** of the outgoing note, without revealing the outgoing B note itself, or to which spent A note the N_A **nullifier** corresponds. More precisely, a plaintext transaction would be formally correct if

1. There is a C_A public **commitment** of a note corresponding to the A note.
2. The N_A **nullifier** is the one corresponding to the A note of the previous C_A **commitment**
3. The entry note has not already been spent (no double spending)
4. The input and output values match (no money creation)
5. The C_B **commitment** of the outgoing note is correctly formed, corresponding to a payment address.

This information is known to the person creating the transaction. The five properties above are verifiable in zero-knowledge mode as follows

1. There is a past public **commitment** corresponding to the entry note, but this **commitment** is kept hidden
2. The revealed **nullifier** corresponds to the note of the previous **commitment**, but this **commitment** is kept hidden
3. The **nullifier** is new
4. Input and output values match but are not revealed
5. The **commitment** C_B of the outgoing note is correctly formed, corresponding to a B note, without revealing the outgoing note.

Only 3. above is checked publicly: a list of already publicised **nullifiers** is maintained. A transaction revealing an already publicly known nullifier is considered a double spend, and will be rejected.

The role of the system (e.g. miners) is to verify the zero-knowledge proof that points 1. 2. 4. and 5. above are true, and to maintain the list of nullifiers to verify 3.

We can see that the **statement** to be proven (the **language** defined by points 1 2 4 5) is defined by the protocol, and that it is always the same during the whole execution of the protocol. The problem of the **trusted setup** only arises once. However, if the protocol should evolve and the validity criteria of a transaction change, the **trusted setup** must be redone. However, **SNARKs** are still preferred because the proofs are extremely short and quick to verify, with relatively little impact on the system.

What are the other costs or issues associated to a zero-knowledge proof system ?

Gilles Fedak : There are two main costs to managing zero-knowledge: the cost of the computations to generate the proofs and the cost associated with setting up the **trusted setups**. These are computations that require a lot of computing and memory resources. They are therefore not at all suitable for blockchains. On the other hand, from the point of view of iExec, which is a distributed computing solution, there could be an opportunity here to use our infrastructure itself to compute proofs or to carry out **trusted setups** in a distributed manner, a bit like the ceremony of Tau.

Anthony Simonet-Boulogne : Indeed, the **trusted setup** for **SNARKs** is an interesting example because it represents a non-negligible cost in both computation time and complexity of implementation. We have learnt that only the person who has done the **trusted setup** will be able to be truly convinced that the **toxic waste** has been eliminated. This person will have confidence in the **structured reference string** because he or she knows that the **trusted setup** was done correctly, but this confidence is difficult to convey. In other words, it is not easy to convince people to trust the reference string and to believe that the **toxic waste** is destroyed.

Improvements to the trusted setup: ceremony of Tau, perpetual ceremony.

Anthony Simonet-Boulogne : From my point of view, we have a centralisation problem with **SNARKs** because their implementation is, in a way, centralised.

Youssef El Housni : That is correct. However, in all applications in production today, the **trusted setup** is done with a Secure multiparty computation (MPC) protocol. Secure multiparty computation protocols allow separate partners to obtain a common result of a certain computation, based on data provided by the participants, without the participants having to share their data with each other. Thus, the **structured reference string** can be constructed by several partners without any of them knowing the secret quantities that form the **toxic waste**.

This was historically done for the crypto-currency **Zcash** in 2016 for the first time and then in 2017 according to the ceremony of Tau. Since then, there have been other MPCs to generate **structured reference strings** for different **statements** (Tornado.cash ceremony, Aztec Ignition, Celo Plumo, filecoin ceremony). These protocols ensure that if only one participant is honest and destroys its secret input, the **toxic waste** cannot be reconstructed by any other participant, or even by a collaborating group.

Anthony Simonet-Boulogne : I would also like to add the notion of a perpetual ceremony of Tau, where **structured reference strings** are constantly being made, strung together. Anyone can add their contribution to the new **structured reference string**, and thus have trust. This makes it possible to join the system and take the keys generated by the previous setups or to add a link in the chain with its own **setup**. If I don't want to benefit from the entropy of the **structured reference string** that others have inserted before me, I can add entropy that I have generated myself, and thus be confident that what I have in my hands is trusted. But once I have done this very costly operation, the newly gained trust only works for me because I cannot easily pass it on to others, except in the form of a promise.

One technology often mentioned when it comes to data protection is the «TEE» (Trusted Execution Environment) which is hardware technology proposed by manufacturers (Intel, AMD, ARM). Why and how does iExec use these technologies?

Gilles Fedak: The term **Trusted Execution Environments** (**TEE**) refers to hardware technology implemented at processor level, for example Intel with SGX enclaves, ARM with TrustZone or AMD with SEV. The idea is that a part of the memory is permanently encrypted and can only be decrypted in the secure enclave. Neither the users nor even the owner of the machine have unencrypted access to what this protected part of memory contains. This technology is interesting for iExec because it ensures the correct and confidential treatment of private data by machines that we do not trust (they are provided on our marketplace). The user encrypts the data before it is circulated on the network and the **TEE** ensures that this data always remains confidential, even when processed after decryption in the **TEE**. This technology is therefore complementary to zero-knowledge, in the sense that it ensures the confidentiality of off-chain data, whereas zero-knowledge ensures the confidentiality of information whose trace is stored on the blockchain.

For us, there are two advantages to using **TEE** technology. The first is that it allows users to encrypt their data so that it can be processed by approved programs, while guaranteeing its confidentiality. This is known as end-to-end encryption. The second advantage is that it is possible to report and record in the blockchain the fact that the result of the execution of an application could not be modified or altered, either by the person who owns the machine on which the application is executed, or by the network, or by anyone else. This makes it possible to transfer to the blockchain a proof of proper execution, of the proper performance of the service for which people have paid.

iExec

iExec is a French technological startup founded in 2016 by two former IT researchers. Based in Lyon, the company currently employs 26 people in the R&D and marketing departments. iExec is developing the first decentralised marketplace allowing individuals or companies to monetise their computing resources. These may be applications such as artificial intelligence algorithms, data sets, or computing resources, during computer downtime for example.

A particularity of iExec is that it financed itself through an ICO (Initial Coin Offering) in April 2017, which raised the equivalent of €11m in Bitcoin. An ICO consists of issuing a new cryptocurrency (in this case the RLC), and exchanging it for existing short-term cryptocurrencies. The RLC token is the exchange medium between the marketplace's stakeholders; it is also a key part of the design of the algorithms that govern the marketplace and implement economic principles to ensure incentive alignment. iExec's mission is therefore to develop technology to address trust, governance and security in the marketplace. To do this, iExec relies mainly on blockchain and confidential computing, including zero-knowledge proofs.

The first users of iExec are blockchain startups as well as large industrial partners. The use cases involve the sharing of confidential data, the use of services in smart-city platforms or the governance of access to user mobility data by a motorway operator.

What is the interest of this technology for iExec?

Anthony Simonet-Boulogne : Zero-knowledge and **rollups** make it possible to find a compromise, this time between performance and centralisation. To enable scaling, most **sidechains** rely on **Proof-of-Authority** protocols which have the effect of more or less «recentralising» consensus by introducing checkpoints. A **Proof-of-Authority** protocol simply consists of identifying authorities that have administrative and validation rights, and therefore control. With zero-knowledge, we hope to achieve similar performance but retain the decentralisation that is so important to us.

Gilles Fedak : iExec is a decentralised marketplace and proposes a protocol to verify that all commercial transactions between stakeholders have taken place at the level of trust they requested when making the deal. This protocol we invented is called «proof of contribution» or PoCo, and is implemented as **smart contracts** on the Ethereum blockchain.

Since this protocol is executed via **smart contracts**, Ethereum miners have to be paid for their execution each time it is triggered. In the Ethereum blockchain, there is a gas mechanism that is consumed to pay for the execution of the **smart contract**, which represents a real financial cost. In addition, the execution of **smart contracts** is relatively slow. So, since this protocol runs on every business interaction, it has a huge impact on performance and therefore for users on usability, user experience, and service quality.

To solve this problem we run our own protocol on a different, faster and cheaper structure than the main blockchain. This is called a **sidechain**. Thanks to this approach, PoCo now runs without any operating costs for users because the gas is free. In addition, bridges form between the main blockchain and our **sidechain**. In the end we are indeed faster and have no running costs, but at the cost of some drawbacks.

The main drawback is that we have somewhat recentralised the whole process in the sense that this **sidechain** is largely administered by iExec. Users ultimately have to trust us to administer and secure the **sidechain**. In addition, the bridges themselves are a centralising factor and a key point of trust: since they are the ones that allow a status to be transmitted from the **sidechain** to the main chain, such as proof that deals have been made and that commercial transactions leading to payments have been carried out. This state allows transactions on the **sidechain** to be translated into valuable transfers on the main chain.

Here, in terms of the trade-off between performance and centralisation, we have focused on performance. We have lost decentralisation but we have gained on economic performance and on performance in terms of transaction throughput.

What we hope is that **rollups** will provide a solution to this problem. One way would be to use **rollups** to link the information in the **sidechain** to the main chain in a secure and again decentralised way. Zero-knowledge could be used to express conditions such as: "Pay me on the main chain because I put in the main chain the proof that in the **sidechain** a commercial transaction took place and that the service was rendered".

Daniel Augot : In conclusion, we can see that the subject of zero-knowledge, which has already been highly industrialised by certain start-ups, is very promising in the world of blockchains. It has the potential to solve crucial industrial problems relating to confidentiality and scaling. The discussion with iExec highlighted the immediate industrial interest of these technologies related to privacy. It is fascinating that an old subject is now opening up great prospects but research is still active to improve the performance of existing systems and to remedy certain defects. The discussion we have just had may be out of date in a year's time, even if standardisation is on the horizon. Stay tuned!

Appendix

Glossary of terms

Educational material

Selectes bibliographic references

Zero-Knowledge low-level computed librairies

Industrial applications

Performances

Standardisation effort

Glossary of terms

To illustrate the notions of zero-knowledge, we use the notion of a compound number, N, with two prime factors P and Q, where N = P* Q

- **Langage**: for a **relation**, it is the set of **instances** that admit a **witness** that satisfies the **relation** between the **instance** and the **witness**. Example: the **language** of the "set of numbers composed of two prime factors"

- **Instance**: public data known to the prover and verifier. "N"

- **Relation**: logical link between an **instance** and a **witness**. "N = P*Q"

- **Statement**: say that an **instance** belongs to a **language**. Example "N is composed of two prime factors": "N" is the **instance** and the **language** is that of numbers composed of two factors.

- **Proof** (or **witness**): data known only to the prover, which establishes that the **instance** belongs to the **language**. Example "P and Q" where N = P*Q

- **Non-interactive zero-knowledge proof**: A string of bytes produced by a prover, which can be verified by the verifier. It must be short, and hide information. It will prove that N = P*Q without revealing P and Q.

- **Backend**: an implementation of cryptographic and low-level mathematical protocols: elliptic curves on finite fields, very long error correcting codes.

- **Frontend**: A way to express **statements** and proofs in a user-friendly computer language, which are then compiled into a low-level mathematical representation suitable for the **backend**.

- **Trapdoor, trapped**: an algorithm is said to be **trapped** if there is a secret quantity, the **trapdoor**, which allows its behaviour or properties to be modified. For example, in a public key encryption system, the **decryption key** is often interpreted as a **trapdoor**: the encryption is impossible to reverse, except for the one who possesses the **trapdoor**, which in this case is the **decryption key**.

- **Encryption key, decryption key**: An encryption algorithm takes a message and an encryption key as input and calculates an encrypted message, which will therefore be unintelligible. A decryption algorithm takes an encrypted message and a **decryption key** as input and calculates the corresponding plaintext message.

- **Cryptographic hashing, hash**: an algorithm that takes arbitrarily long messages as input, and calculates a very short **hash** (typically 32 bytes). This algorithm has the property of making it impossible, given a **hash**, to find the input message.

- **Commitment**: publishing or sharing a document **hash**, without revealing the document. For example, the document **hash** can be stored in a blockchain, instead of the document.

- Opening a **commitment**: given a **commitment** of a document or data, the document is revealed and compared with its **hash**. It is verified that the document has been pledged. For example, the **commitment** can be recorded in a blockchain. The owner of the document can then use the block recording the **hash** as the date of registration and cannot change the document once it has been pledged. This is a kind of blind notarisation. Here, however, zero-knowledge evidence can prove the validity of **statements** about a document committed by its **hash** without revealing it.

- **Merkle tree**: an aggregative data structure for pledging a batch of a large number of documents using successive **hashes**. It has two interesting properties. The root **hash** is that of all the documents and remains very short. Second, it is possible for the **Merkle** tree manager to prove at low cost that a document is part of the batch, without revealing the other documents.

- **Oracle**: A blockchain can only certify the data already recorded in the blockchain and the transactions made on it. A blockchain in itself does not guarantee the quality of external data inserted. For example, if **smart contracts** depend on a local micro weather forecast, one has to trust the operator who enters the weather data to record it in the blockchain. The term **oracle** is used to designate such an operator and to emphasise the notion of trust in external data.

- **SideChain**: this is a blockchain set up in parallel with the main chain for reasons of performance and confidentiality, with validation and consensus mechanisms that may be different from those of the main chain. This raises the problem of the compatibility and reconciliation of transactions executed in the two chains.

- **ZK-Rollup**: a transaction batch is generated outside the main blockchain. This batch is recorded in a **Merkle** tree, of which only the root is recorded in the main chain. This root is accompanied by a short zero-knowledge proof that all transactions in the batch are correct.

- **Proof-of-authority**: this is simply an electronic signature with the **private key** of a well-identified entity, in which the system or part of the system trusts and is able to verify the signatures issued by that entity. This allows the validation of transactions in private or permissioned blockchains, but is antagonistic to the notions of public blockchains and decentralisation.

- **Trusted setup**: The process by which a **ZK-SNARK** is initialised. This process produces a **structured reference string**.

- **Structured reference string, toxic waste**: In the context of **ZK-SNARKs**, which allow very short proofs to be verified quickly, all the complexity is pushed into a byte string, which encodes the algebraic steps that must be performed to verify a zero-knowledge proof. This string is set up at system initialisation, for a given **language**. This string, built according to the principles of public key cryptography, is **trapped**. Whoever knows about the **trapdoor** can then make proofs that are accepted as correct of false **statements**, hence the common expression "**toxic waste**" to refer to the **trapdoor**.

- **Smart contract**: a computer program stored in the blockchain. This program is executed by validators or miners when transactions trigger it. A **smart contract** typically allows the deployment of its own application logic on top of the blockchain that holds it, for example the issue of fungible or non-fungible tokens, or certificates for digital assets, etc. In addition, a **smart contract** allows purchases or sales between tokens and the blockchain's native currency.

- **Trusted Environment Execution**: hardware technology implemented at the processor level (Intel SGX, ARM TrustZone with TrustZone, AMD SEV). This is a part of the memory that is permanently encrypted and can only be decrypted in the secure enclave, according to the manufacturer's guarantees. In addition, a calculation can be performed on the data in the enclave and output the result, with the guarantee that the calculation is correct. This guarantee is evidenced by a publicly verifiable electronic signature.

Educational materials

- **A simple and clear presentation**: *How to Explain Zero-Knowledge Protocols to Your Children*. Quisquater, Jean-Jacques ; Guillou, Louis C. ; Berson, Thomas A. CRYPTO `89. [online] http://www.cs.wisc.edu/~mkowalcz/628.pdf

- **Compiling a program into a system of polynomials** : https://electriccoin.co/blog/snark-explain5/

- **Step by step presentation of the cryptographic mechanisms of a SNARK**: https://electriccoin.co/blog/snark-explain/

- Zokrates, an environment for building SNARKs and associated **smart contracts**: https://zokrates.github.io/

- **Step-by-step process of a STARK based on an example**: https://starkware.co/developers-community/stark101-onlinecourse/

Selected bibliographic references

- **Invention of zero-knowledge**: *The knowledge complexity of interactive proof systems*. Goldwasser, S., Micali, S., Rackoff, C. (1989). SIAM Journal on Computing, 18 (1) : 186–208. [online] http://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Proof%20Systems/The_Knowledge_Complexity_Of_Interactive_Proof_Systems.pdf

- **Seminal paper on proofs based on elliptic curves and pairings**: *Short Pairing-Based Non-interactive Zero-Knowledge Arguments*. Jens Groth, ASIACRYPT 2010. [online] https://www.iacr.org/archive/asiacrypt2010/6477323/6477323.pdf

- **Introduction of the SNARK term**: *From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again*. Nir Bitansky and Ran Canetti and Alessandro Chiesa and Eran Tromer, ICTS'12. [online] https://eprint.iacr.org/2011/443

- **First efficient SNARK**: *Pinocchio: Nearly Practical Verifiable Computation*. Bryan Parno and Craig Gentry and Jon Howell and Mariana Raykova, IEEE Symposium on Security & Privacy 2013. [online] https://eprint.iacr.org/2013/279

- **Invention of STARKs**: *Scalable, transparent, and post-quantum secure computational integrity*. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, Michael Riabzev. March 6, 2018 [online] https://eprint.iacr.org/2018/046.pdf. Simplified version by the same authors: Scalable Zero Knowledge with no Trusted Setup, CRYPTO 2019 [by subscription]

Zero-knowledge low-level computer libraries

- **libsnark** (Scipr-lab.org association of academics)
- **gnark** (Consensys)
- **starkware-libs** (Starkware)
- **Bellman** (as used in zcash)

Industrial applications

- **Zcash**
- **StarkEx**: rollup in production
- **Aztec.network** (zk.money, rollup PLONK)
- **zkSync** (ZK rollup in production)
- **Filecoin** : peer-to-peer storage system in exchange for payment

Performances

- ZK-SNARKs for **Zcash** : 130,000 gate circuits as a matter of routine
- **Filecoin** requires a proof system for a circuit with nearly a billion gates.
- **Starks** : 3 000 transactions per second
  https://medium.com/starkware/the-great-reddit-bake-off-2020-c93196bad9ce
- Example outside blockchain: systems pushed to the extreme by **DARPA** (Defense Advanced Research Projects Agency). This involves proving the existence of a software flaw in a program without revealing the flaw in question. The proof can be gigantic in this case.

Standardisation effort

- Zkproof.org

Daniel Augot

Director of research at Inria (French National Institute for Research in Computer Science and Control), he obtained a thesis in computer science in 1993 and the accreditation to direct research in 2007. He supervises doctoral and post-doctoral students at the École Polytechnique. With Julien Prat, he is responsible for the Blockchain and B2B platform chair, supported by CapGemini, NomadicLabs, and the Caisse des dépôts.

Sarah Bordage

A doctoral student since 2018 at École polytechnique under the supervision of Daniel Augot. She is working on zero-knowledge proof constructions for verifiable computation.

Youssef El Housni

Engineer at ConsenSys, member of the «gnark» team and PhD student at École polytechnique under the supervision of Daniel Augot and François Morain. He is working on zkSNARKs proofs and the underlying cryptographic primitives, in algorithmic number theory.

Xavier Lavayssière

Independent researcher on technological and regulatory aspects of digital assets. He founded the ECAN, https://ecan.fr/. training centre on blockchain technologies and teaches at Paris I Panthéon Sorbonne.

Louis Bertucci

Researcher at the Louis Bachelier Institute. He obtained a PhD in finance from the University of Paris-Dauphine. He has been working on blockchains since 2017..

Noémie Dié

Doctoral student at the Economics Department of Télécom Paris (Institut Polytechnique Paris) in partnership with Bpifrance Le Lab.

Gilles Fedak

PhD in computer science from the University of Paris Sud. After a post-doctorate at the University of California in San Diego, he became an INRIA researcher at ENS Lyon. He is the recipient of the Chinese PIFI prize. Gilles Fedak is CEO and founder of iExec, a cloud computing platform based on blockchains..

Anthony Simonet-Boulogne

PhD in Computer Science from the École Normale Supérieure de Lyon in 2015. He has worked on problems related to distributed computing, cloud computing and blockchain at Inria, Rutgers University and has been Scientific Project Manager at iExec since 2019.

BLOCKCHAIN&
PLATFORM CHAIR

l'X ÉCOLE POLYTECHNIQUE

INSTITUT POLYTECHNIQUE DE PARIS

Capgemini

nomadic labs

Caisse des Dépôts GROUPE

Director of publication: **Daniel Augot**

With the kind participation of:

Sarah Bordage	Gilles Fedak
Youssef El Housni	Anthony Simonet-Boulogne

And the proofreading assistance of:

Louis Bertucci	Xavier Lavayssière

Editorial assistance: **Noémie Dié**

Graphic design and layout: Centre Polymédia de l'Ecole Polytechnique
Outlook publication of the Polytechnique Blockchain&Platform Chair
Capgemini, CDC, NomadicLab. All rights reserved