New Privacy Practices for Blockchain Software

Marta Bellés-Muñoz,[1,2] Jordi Baylina,[1] Vanesa Daza,[2] and José L. Muñoz-Tapia[3]
[1] *Iden3*
[2] *Universitat Pompeu Fabra*
[3] *Universitat Politècnica de Catalunya*

The benefits of blockchain technologies for industrial applications are unquestionable. However, it is a considerable challenge to use a transparent system like blockchain and at the same time provide privacy to sensitive data. Privacy technologies permit conducting private transactions about sensitive data over transparent networks, but their inherent complexity has been overwhelming for many developers. Closing the gap between developers and privacy-preserving technologies would help to the full adoption of the privacy by design framework for blockchain software. To this end, in this paper we present the software tools we have implemented to bring complex privacy technologies closer to developers and facilitate the job of implementing privacy-enabled blockchain applications.

INTRODUCTION

Novel decentralized business processes are arising thanks to blockchain technologies. A remarkable scenario that can be better managed using blockchain is the new smart manufacturing paradigm that uses on-demand access to a shared collection of diversified and distributed manufacturing resources [1]. In this context, the use of blockchain can bring in significant business benefits, such as greater transparency, improved traceability, enhanced security, better reconciliation processes, increased transaction speed, and costs reduction [2].

While the public transparency of blockchains is a desirable feature, it becomes a concern when dealing with privacy. Although it may seem that guaranteeing public transparency and privacy is incompatible, it is actually possible with a cryptographic technique called Zero-Knowledge Proof (ZKP). ZKPs are very powerful tools for implementing applications with sophisticated privacy requirements. However, these tools have an inherent complexity that has been overwhelming for many developers. For this reason, many times developers did not participate in the design of the application's privacy, and it was mainly done by cryptographers.

Having developers and cryptographers without a common language is prone to cause misunderstandings. To some extent, the situation is similar to what happened between operation and development teams. The appearance of DevOps practices that were able to connect the two worlds not only reduced the frictions but contributed to speed up the whole software development process. In our opinion, an analogous situation could happen here.

In this paper, we present a novel framework that closes the gap between ZKP technology and developers. We describe the set of tools we have developed to express privacy requirements in a friendly way. We also provide links to guided tutorials of the different parts of the process as well as the repositories of our software.

ZERO-KNOWLEDGE PROOFS

A ZKP is a protocol that enables one party, called prover, to convince another, called verifier, that a statement about certain data is true, without leaking or disclosing any extra information beyond the veracity of the statement. That is, with ZKPs it is possible to prove the veracity of logic statements that concern private data. For example, a prover can create proofs for statements like the following:

- *"I know the private key that corresponds to this public key"*: in this case, the proof would not reveal any information about the private key.
- *"I know the preimage of this hash value"*: in this case, the proof would show that the prover knows the preimage but it would not reveal any information about the value of that preimage.
- *"This transaction privately transfers a coin"*, in this case, the proof would not reveal any information about the origin, destination or amount being transferred but it would still ensure that the coin has not been double spent.

In the past, ZKP systems were heavily coupled to the statements being proved and changing a statement required a redesign of the cryptographic system and a new security analysis. As a result, modifying any privacy statement was hard to analyse and implement by non-cryptographers. The appearance of modern ZKPs shifted the paradigm by decoupling the statement definition from the proving mechanism. With modern ZKPs, statements can be defined as arithmetic circuits and the proof can be generated from the circuit definition.

CIRCUITS

Arithmetic circuits are circuits built connecting wires that carry elements from a large prime field to addition

and multiplication gates. For example, a NAND gate can be implemented with an arithmetic circuit, as shown in Fig. 1.

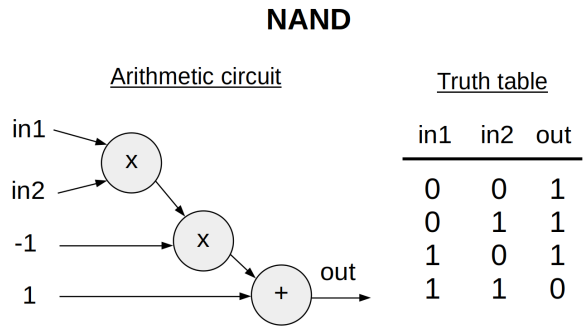

FIG. 1: The NAND gate implemented with an arithmetic circuit: `out = 1 - in1 x in2`.

This example is important because the NAND is a universal gate, meaning that any other logic gate can be represented as a combination of NAND gates. As a result, any computation that a computer can perform can be implemented with a combination of connected NANDs. Therefore, since a NAND can be implemented with an arithmetic circuit, with this type of circuits we can perform the same complex computations a computer can do, like calculating a hash or verifying a digital signature.

The main feature of circuit-based ZKPs is that they allow the prover to generate a proof that shows that she has an assignment of inputs that produces a predefined output, while keeping part or all the inputs secret. In particular, proofs have two fundamental properties: (i) a prover cannot generate a valid proof without knowing correct inputs for the circuit, and (ii) proofs securely obfuscate the private inputs, meaning that they do not leak information about the private inputs to the verifier.

Among modern ZKP-circuit based protocols, the most interesting ones for blockchain are ZK-SNARKs (Zero-knowledge Succinct Non Interactive ARgument of Knowledge), because verification is not intensive in terms of bandwidth and computing, and they can be implemented in a blockchain smart contract [3]. In particular, ZK-SNARK proofs are always small ($\approx$ 200 bytes), regardless of the size of the circuit, and the time to verify a proof is also short and constant [4].

ZK-SNARKS

As shown in Fig. 2, the first step for creating an application powered by ZK-SNARKs is the definition of a circuit. An option is to provide this definition in the form of connected addition and multiplication gates (low-level circuit description). However, for complex circuits, writing such a low-level description implies dealing with millions of gates, becoming an extremely laborious and error-prone task. A more practical option is to use a tool (a circuit compiler) that allows developers to provide high-level circuit definitions that can be compiled to their corresponding low-level descriptions.

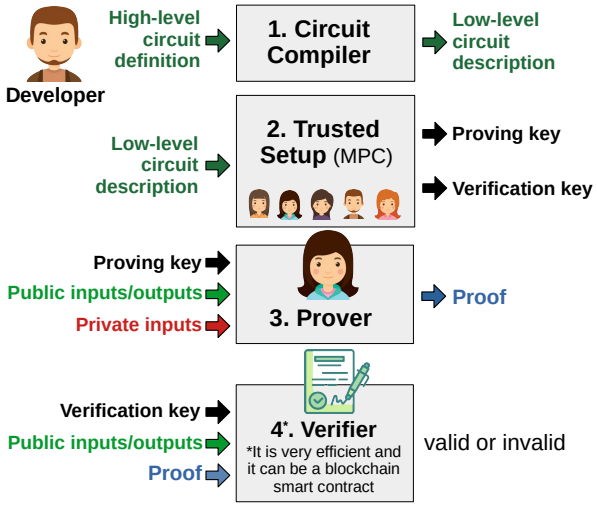

FIG. 2: Overview of how ZK-SNARKs work.

At this point, we must remark that the low-level description of a circuit is not enough to use a ZK-SNARK. A second step called trusted setup (TS) [4] is also needed. In the TS, a trusted entity produces the cryptographic material needed by the prover (proving key) and by the verifier (verification key). The entity that runs the TS is trusted because it needs to generate and then destroy certain random values. In fact, if the random values are ever exposed, the security of the whole scheme is compromised. Centralization in this delicate setup phase can be avoided by means of a multi-party computation (MPC) [5]. MPC allows multiple independent parties to collaboratively construct the TS parameters and, in this process, it is enough that one single participant deletes its secret counterpart of the contribution to keep the whole scheme secure.

With the proving key, the prover can create proofs using public and private values (step three), and with the verification key, the verifier can later verify these proofs (step four). It is important to remark that it is always possible to check that the proving and verification keys derive from a certain circuit and MPC.

APPLICATIONS

The importance of efficient circuit-based ZKPs for blockchain is two-fold. On the one hand, they allow efficient implementations of privacy requirements, and on the other, their small proof size and verification time can be used for enhancing the scalability of blockchains.

Regarding privacy, ZKPs can be applied to enhance authentication processes, as we will discuss using an example in the context of a smart factory. ZKPs can also be used to improve the privacy of payments. Examples are the Zcash blockchain [6] and Tornado Cash [7], which is implemented over the Ethereum public network using our software. Implementations of legal compliance and controlled disclosure of personal information are also important applications of ZKPs. For example, one can prove to a bank that earns above a certain minimum amount that is required to repay a loan, without revealing the actual salary.

The other big area where ZKPs are being applied to blockchain is for improving the network scalability. In this context, the emergence of Decentralized Finance (DeFi) has put pressure to increase the number of transactions per second that a blockchain can support. Many blockchains are using the concept of ZK-Rollups to enhance scalability. The idea of a ZK-Rollup is to use mass transfer processing rolled into a single transaction and check the correctness of the final state with a zero-knowledge proof. Hermez Network [8] is a project using our software to implement a ZK-Rollup over Ethereum.

OUR SOFTWARE CONTRIBUTIONS

At iden3, in collaboration with the Ethereum foundation [9] and the research teams of several universities, we have developed a new language for describing circuits, its associated compiler called CIRCOM [10], and a crypto-compiler called SNARKJS [11] for generating and verifying ZK-SNARK proofs as defined in [4].

There are other solutions that allow the generation and verification of zero-knowledge proofs, such as `LibSnark` (written in C++, developed by SCIPR Lab), `ZoKrates` (in Python, by TU Berlin), `Bellman` (in Rust, by Zcash), `Snarky` (in OCaml, by o1-labs), `Zinc` (in Rust, by MatterLabs), `WasmSnark` (in WebAssembly, by iden3). The main contribution and novelty of our framework is that:

- It decouples the circuit definition (CIRCOM) from the proving system (SNARKJS). This makes it easier to update and integrate other crypto-compilers.
- It covers the MPC generation and verification processes, providing a holistic framework for building privacy-enabled applications.

CIRCOM is a developer-friendly language that makes it possible to create large circuits by combining smaller ones called *templates*. We provide a library of templates called CIRCOMLIB [12] with hundreds of circuits such as comparators, hash functions, digital signatures, binary and decimal convertors, and many more. The idea of building large and complex circuits from small individual components makes it easier to test, review and audit the resulting code. In this regard, CIRCOM users can create their own custom templates, but using circuits from the library has the advantage that they have been reviewed by our research team and the open-source community supporting the project. Once a circuit is created in CIRCOM, SNARKJS makes the proof generation and the proof validation automatic and transparent.

We would like to remark that the library, the language and both compilers are open source and publicly available to practitioners and developers. In the following section, we use an illustrative example in the context of a distributed smart factory to show how the whole process works.

A DISTRIBUTED SMART FACTORY

To illustrate the process of building a blockchain application with privacy powered by ZK-SNARKs, we consider a smart contract deployed in a blockchain that rules the production of a distributed smart factory. When the smart contract receives an appropriate transaction, it registers the action "start fabricating a certain amount of products". Then, systems in the factory, which look for such registrations in the blockchain, automatically start the corresponding fabrication processes.

In this context, consider that transactions with orders for approving fabrication can be given by any of two authorized parties, denoted by A and B. As a privacy requirement, we want to avoid that competitors learn who is giving a particular order in the smart factory. Note that public blockchains are transparent, so sending regular transactions is not adequate, since anyone could deduce the identity of the party approving the order just by looking at the transaction's signature. In this section, we show how to use ZK-SNARKs to keep the signer's identity hidden making use of our framework.

Circuit Design

We want parties A and B to be able to interact with the smart contract ruling the smart factory, without revealing which of the two identities is giving an order. As shown in Fig. 3, instead of using regular signed transactions, A and B can send a transaction that includes a proof that shows that an authorized public key signed the order. That is, given a fabrication order, we want A and B to be able to generate a proof for the statement

"this fabrication order is signed by one of the two authorized public keys (pk_A or pk_B)",

without revealing the signer's identity. In this case, we need a circuit that, given two authorized public keys, a message (the fabrication order) and a signature, checks if the signature is valid for one of the two public keys.

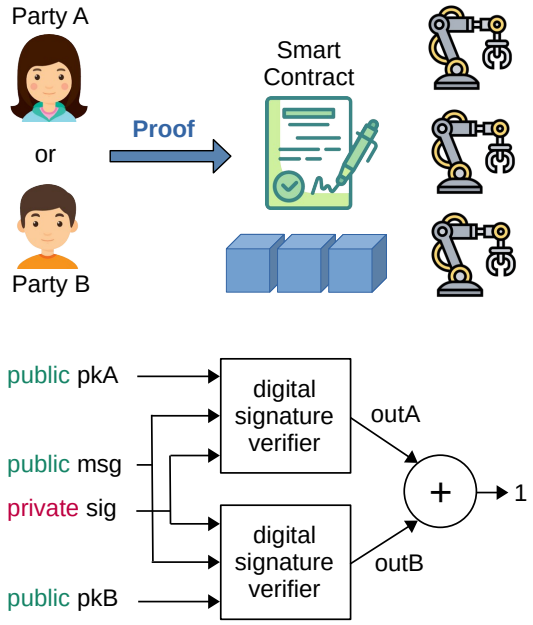

FIG. 3: Authorized parties A and B can send a transaction to the blockchain smart contract with a proof showing that the fabrication order was emitted by one of them, but without revealing who.

The circuit from Fig. 3 checks if a signature `sig` on a message `msg` corresponds to a signature emitted by one of two authorized public keys, `pkA` or `pkB`. Each `digital signature verifier` box represents a subcircuit that combines addition and multiplication gates to check if `sig` is a valid digital signature on `msg` by a given public key. That is, given `pk`, the `digital signature verifier` template outputs `1` if `sig` is a valid signature on `msg` for `pk`, and `0` otherwise. As a result, the circuit is satisfied if either `outA = 1` or `outB = 1`. Equivalently, if we assume that `pkA` and `pkB` are different, the circuit is satisfied if and only if

```
outA + outB = 1.
```

A valid proof on this circuit would demonstrate that the prover has the correct inputs that satisfy the circuit but would not reveal any information about the private input `sig` (the signature). This way, an authorized party can show that she holds a valid public key to sign the fabrication order but keep her signature secret. Next, we explain how to implement the circuit with CIRCOM.

High-level Definition with CIRCOM

The mechanism to define circuits in CIRCOM is through the use of templates. The instantiation of a template called *component*. We implement the circuit from Fig. 3 in a template named `AuthorizeFabricationOrder`, and instantiate it in the `main` component.

```
include "eddsa.circom";

template AuthorizeFabricationOrder() {
    signal public input pkA;
    signal public input pkB;
    signal public input msg;
    signal private input sig;

    signal outA;
    signal outB;

    component verifyA = EdDSAVerifier();
    component verifyB = EdDSAVerifier();

    // verify signature with pkA
    verifyA.pk <== pkA;
    verifyA.msg <== msg;
    verifyA.sig <== sig;
    outA <== verifyA.out;

    // verify signature with pkB
    verifyB.pk <== pkB;
    verifyB.msg <== msg;
    verifyB.sig <== sig;
    outB <== verifyB.out;

    outA + outB === 1;
}

component main = AutorizeFabricationOrder();
```

Arithmetic circuits implemented with CIRCOM operate on signals, which are declared using the keyword `signal`. An `input` of a circuit is a particular type of `signal` and it can be `public` or `private`. `AuthorizeFabricationOrder` has three `public inputs` (the two authorized public keys and the message) and one `private input` (the signature).

For the implementation of the `digital signature verifier` box, we use a template called `EdDSAVerifier` that is imported from CIRCOMLIB. Note that the templating mechanism provided by CIRCOM allows us to use the `EdDSAVerifier` template as a black box that returns a signal that determines if a signature [13] is valid for a given message and public key. The `EdDSAVerifier` template is defined in the file `eddsa.circom`, which is included in our circuit using the keyword `include`. Notice that since we need to verify the signature twice, one per key, the template `EdDSAVerifier` is instantiated in two different components: `verifyA` and `verifyB`.

The expression `outA + outB === 1`, imposes the constraint that the circuit `AutorizeFabricationOrder` is satisfied only if the output of `verifyA` or `verifyB` is `1`.

With the high-level circuit definition, a developer can download our circuit compiler (CIRCOM) and our crypto-compiler (SNARKJS) from our repositories and follow the tutorial in [11] to run an MPC, generate the proving and verification keys and finally, create and verify proofs. However, to make this process even easier and facilitate collaboration, we have created a service called CIRCOMHUB. As we explain in the following section, CIRCOMHUB hosts circuit definitions and their associated cryptographic material, making it easy to share data between developers, provers and verifiers.

Trusted Setup, Proof Generation and Proof Verification

As shown in Fig. 4, developers can upload their high-level circuit definitions to CircomHub. Then, CircomHub compiles the defintion with circom and passes the result (the low-level circuit description) to the snarkjs compiler.

snarkjs provides an MPC module that allows multiple independent parties to collaboratively construct the trusted setup following the protocol from [5]. CircomHub acts as a cloud service that facilitates the storage of the different phases of the MPC.

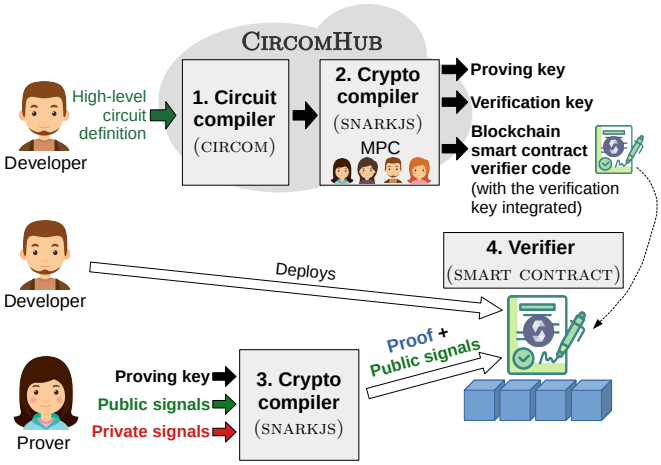

FIG. 4: Our framework tools.

The snarkjs compiler uses the circom circuit definition and the MPC result to generate the cryptographic material for the prover and the verifier (step 2). In addition, the crypto-compiler also generates smart contract code for proof verification on blockchain. More precisely, the compiler generates Solidity code, which is one of the most widely used languages for writing smart contracts. All this information is stored in CircomHub and made available through a public API. It is important to remark that CircomHub acts as a repository and not as a trusted party, since everything hosted there (circuit, compilation of the circuit, proving and verification keys) is publicly available and verifiable according to [5].

Developers can use the on-chain verifier generated by snarkjs to deploy a smart contract in the blockchain that integrates a ZK-SNARK verifier. This way, it is a smart contract (step 4) who verifies if a proof included in a transaction is valid according to [4].

On the other hand, an authorized party can use snarkjs and the proving key stored in CircomHub to generate a proof. snarkjs will use the proving key, the authorized public keys (public inputs), the fabrication order (public input) and a valid signature (private input) to generate a proof (step 3).

Finally, the prover can include this proof in the fabrication order transaction, which will only take place if the verifier in the smart contract accepts the proof.

There is still one issue related to privacy that needs to be solved. Transactions in blockchain need to be signed by the sender, so if the prover signs the transaction she would lose her anonymity. To obfuscate the member's identity, a simple solution is to use relayers. In this case, the prover sends the proof to a relayer, who signs and sends the transaction to the blockchain. Relayers do not need to be trusted, since it is not possible for them to generate fake proofs. An example of a relayer network for the Ethereum blockchain is the Gas Station Network (GSN), which is a decentralized network of relayers [14].

CONCLUDING REMARKS

Developer-friendly languages like circom, that allows us to express privacy requirements in a friendly way, will bring complex privacy technologies closer to development. We would like to encourage interested readers to follow our guided tutorials to get familiar with our tools and explore the possibilities they offer.

ACKNOWLEDGEMENTS

This research is supported by the Ethereum Foundation Ecosystem Support [9], TCO-RISEBLOCK (PID2019-110224RB-I00), H2020-i3-MARKET, ARPASAT (TEC2015-70197-R), 2014-SGR-1504, RTI2018-102112-B-I00 (AEI/FEDER,UE) and H2020 PRESENT (856879).

[1] R. G. Durán, et al., "An Architecture for Easy Onboarding and Key Life-Cycle Management in Blockchain Applications", *IEEE Access*, vol. 8, pp. 115005-115016, (2020).

[2] V. J. Morkunas, et al., "How Blockchain Technologies Impact Your Business Model", *Business Horizons*, vol. 62, issue 3, pp. 295-306, (2019).

[3] Z. Zheng, et al., "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends", *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 557-564, Honolulu, HI, (2017).

[4] J. Groth, "On the Size of Pairing-based Non-interactive Arguments", *Advances in Cryptology – EUROCRYPT 2016*. LNCS, vol 9666, pp. 305-326, Springer, Berlin, Heidelberg, (2016).

[5] S. Bowe and A. Gabizon, "Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model", *IACR Cryptology ePrint Archive*, Report 2017/1050, (2017).

[6] E. Ben-Sasson, et al., "Zerocash: Decentralized Anonymous Payments from Bitcoin", *2014 IEEE Symposium on Security and Privacy*, pp. 459-474, San Jose, CA, (2014).

[7] D. Khovratovich and M. Vladimirov, "Tornado Privacy Solution. Cryptographic Review. Version 1.1.", `https://tornado.cash/audits/TornadoCash_cryptographic_review_ABDK.pdf`, (2019).
[8] Hermez Network, "Hermez Whitepaper", (October 2020), `https://hermez.io/hermez-whitepaper.pdf`, accessed: 2020-12-29.
[9] Ethereum Foundation Ecosystem Support, "CIRCOM Featured Project", (2020), `https://esp.ethereum.foundation/en/projects/circom/`, accessed: 2020-12-29.
[10] "CIRCOM: Circuit Compiler for Zero-Knowledge Proofs", (2020), `https://github.com/iden3/circom`, accessed: 2020-12-29.
[11] "SNARKJS: JavaScript Implementation of zk-SNARKs", (2020), `https://github.com/iden3/snarkjs`, accessed: 2020-12-29.
[12] "CIRCOMLIB: Library of Circom Templates", (2020), `https://github.com/iden3/circomlib`, accessed: 2020-12-29.
[13] S. Josefsson and I. Liusvaara, " Edwards-Curve Digital Signature Algorithm (EdDSA)", *IRTF*, RFC: 8032, (2017).
[14] "GSN Ethereum Gas Station Network", `https://www.opengsn.org/`, accessed: 2020-12-29.

Marta Bellés-Muñoz is a PhD student doing research in security and efficiency of arithmetic circuits for zero-knowledge protocols at Pompeu Fabra University in collaboration with iden3. Contact: marta.belles@upf.edu.

Jordi Baylina is one of the most outstanding members of Ethereum community and part of the White Hat Group. He is the co-founder of iden3 and the main contributor to CIRCOM and SNARKJS. Contact: jordi@baylina.cat.

Vanesa Daza is an Associate Professor at Pompeu Fabra University. Her main research interests are distributed cryptographic techniques to improve security and privacy of blockchains. Contact: vanesa.daza@upf.edu.

José L. Muñoz-Tapia is an Associate Professor and director of the Master in Blockchain Tehcnologies at Polytechnic University of Catalonia. His research interests are applied cryptography and distributed ledgers. Contact: jose.luis.munoz@upc.edu.