New Privacy Practices for Blockchain Software

Marta Bellés-Muñoz, Iden3 and Universitat Pompeu Fabra

Jordi Baylina, Iden3

Vanesa Daza, Universitat Pompeu Fabra

José L. Muñoz-Tapia, Universitat Politècnica de Catalunya

// In this article, we present the software tools we have implemented to bring complex privacy technologies closer to developers and to facilitate the implementation of privacy-enabled blockchain applications. //

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 cost reductions.[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 guaranteeing privacy are incompatible, it is actually possible to guarantee both with a cryptographic technique called *zero-knowledge proofs* (*ZKPs*). 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, developers often did not participate in the design of an application's privacy; instead it was mainly done by cryptographers.

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

In this article, 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 to the repositories of our software.

ZKPs

A ZKP is a protocol that enables one party (called the *prover*) to convince another (called the *verifier*) that a

Digital Object Identifier 10.1109/MS.2021.3086718
Date of current version: 18 April 2022

statement about certain data is true without leaking or disclosing any 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 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 would still ensure that the coin had 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, the modification of any privacy statement was hard to analyze and implement by noncryptographers. 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 built connecting wires that carry elements from a large prime field to addition and multiplication gates. For example, a NOT-AND (NAND) gate can be implemented with an arithmetic circuit, as presented in Figure 1.

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, by making combinations of this type of circuit, we can perform the same complex computations as a computer, 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 he/she has an assignment of inputs that produce a predefined output while keeping part of or all of the inputs secret. In particular, proofs have two fundamental properties: 1) a prover cannot generate a valid proof without knowing the correct inputs for a circuit, and 2) proofs securely obfuscate the private inputs, meaning that they do not leak information about them to the verifier.

Among modern ZKP-circuit-based protocols, the most interesting ones for blockchain are zero-knowledge succinct noninteractive argument of knowledges (ZK-SNARKs) because their 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 (≈ 200 bytes) regardless of the size of the circuit, and the time to verify a proof is short and constant.[4]

ZK-SNARKs

As displayed in Figure 2, the first step for creating an application powered by ZK-SNARKs is defining a circuit. One 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 would require dealing with millions of gates, making the defining 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.

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 the *trusted setup* (*TS*)[4] is also needed. In the TS, a trusted entity produces the cryptographic material needed by the prover (the proving key) and by the verifier (the verification key). The entity that runs the TS is

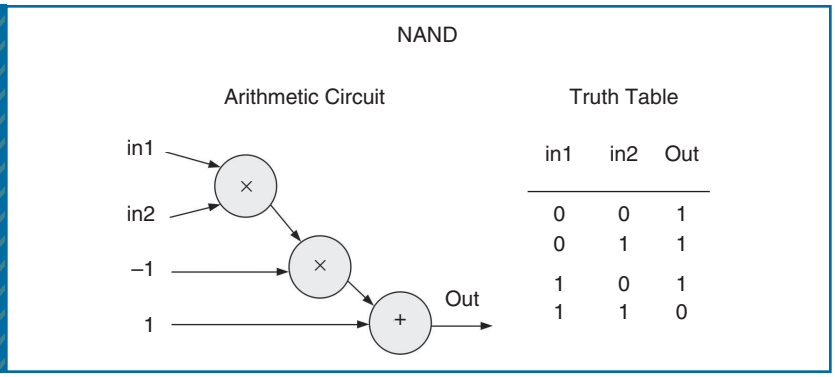

FIGURE 1. The NAND gate implemented with an arithmetic circuit: out = 1 – in1 x in2.

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 MPC.[5] MPC allows multiple independent parties to collaboratively construct the TS parameters. In this process, one single participant deleting its secret counterpart of the contribution is enough 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 include the Zcash blockchain[6] and Tornado Cash,[7] which is implemented over the Ethereum public network using our software. Implementations of legal compliance and the controlled disclosure of personal information are also important applications of ZKPs. For example, one can prove to a bank that he/she earns above a certain minimum amount required to repay a loan without revealing his/her actual salary.

The other big area in which ZKPs are being applied to blockchain is for improving network scalability. In this context, the emergence of decentralized finance has generated pressure to increase the number of transactions per second that a blockchain can support. Many blockchains are using ZK-Rollups to enhance scalability. The idea of a ZK-Rollup is to use mass-transfer processing rolled into a single transaction and to check the correctness of the final state with a ZKP. 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 cryptocompiler called *snarkjs*[11] for generating and verifying ZK-SNARK proofs as defined Groth.[4] There are other solutions that allow the generation and verification of ZKPs, 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), and **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 cryptocompilers.
- 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

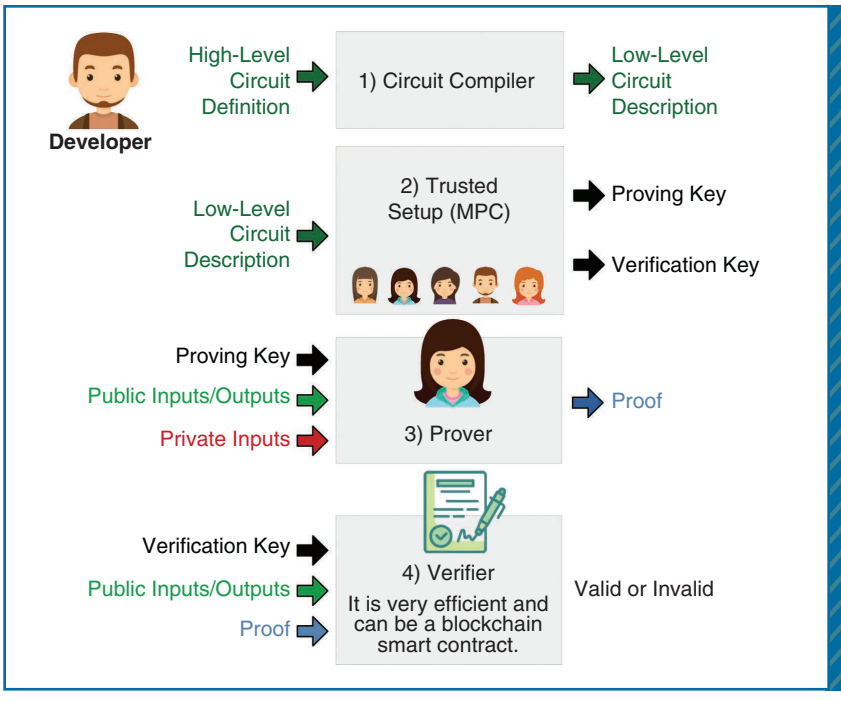

FIGURE 2. An overview of how ZK-SNARKs work. MPC: multiparty computation.

large circuits by combining smaller ones called *templates*. We provide a library of templates, circomlib,[12] with hundreds of circuits, including comparators, hash functions, digital signatures, binary and decimal convertors, and many more. 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. However, using circuits from the library is advantageous because 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 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 that the action should 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 competitors learning 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 the next section, making use of our framework, we present how to use ZK-SNARKs to keep a signer's identity hidden.

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 is giving an order. As presented in Figure 3, instead of using a regular signed transaction, 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.

The circuit from Figure 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**,

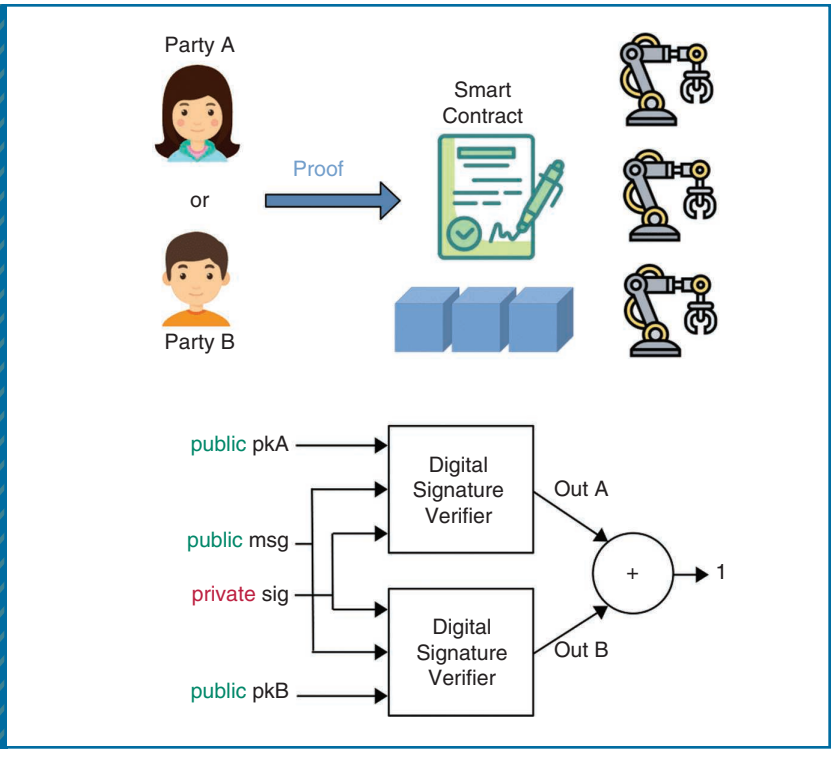

FIGURE 3. The authorized parties A and B can send a transaction to the blockchain smart contract with a proof showing that the fabrication order was emitted by one of them but without revealing who.

the 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 to satisfy the circuit but would not reveal any information about the private input sig. This way, an authorized party can show that he/she holds a valid public key to sign the fabrication order while keeping his/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 is called the *component*. We implement the circuit from Figure 3 in a template, 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 =
  AuthorizeFabricationOrder();
```

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 the EdDSAVerifier template, 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 eddsa.circom file, which is included in our circuit using the keyword "include." Notice that since we need to verify the signature twice (one per key), the EdDSAVerifier template is instantiated in two different components: verifyA and verifyB. The expression outA + outB === 1 imposes the constraint that the circuit AuthorizeFabricationOrder is satisfied only if the output of verifyA or verifyB is 1.

With the high-level circuit definition, a developer can download circom and snarkjs from our repositories and follow the tutorial in "snarkjs: JavaScript Implementation of zk-SNARKs"[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 among developers, provers, and verifiers.

TS, Proof Generation, and Proof Verification

As displayed in Figure 4, developers can upload their high-level circuit definitions to CircomHub. Then, CircomHub compiles the definition 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 TS following the protocol from Bowe and Gabizon.[5] CircomHub acts as a cloud service that facilitates the storage of the different phases of the MPC.

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 cryptocompiler 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 of this information is stored in CircomHub and made available through a public application programming interface. 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 Bowe and Gabizon.[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, the smart contract (step 4) verifies if a proof included in a transaction is valid according to Groth.[4]

However, an authorized party can use snarkjs and the proving key

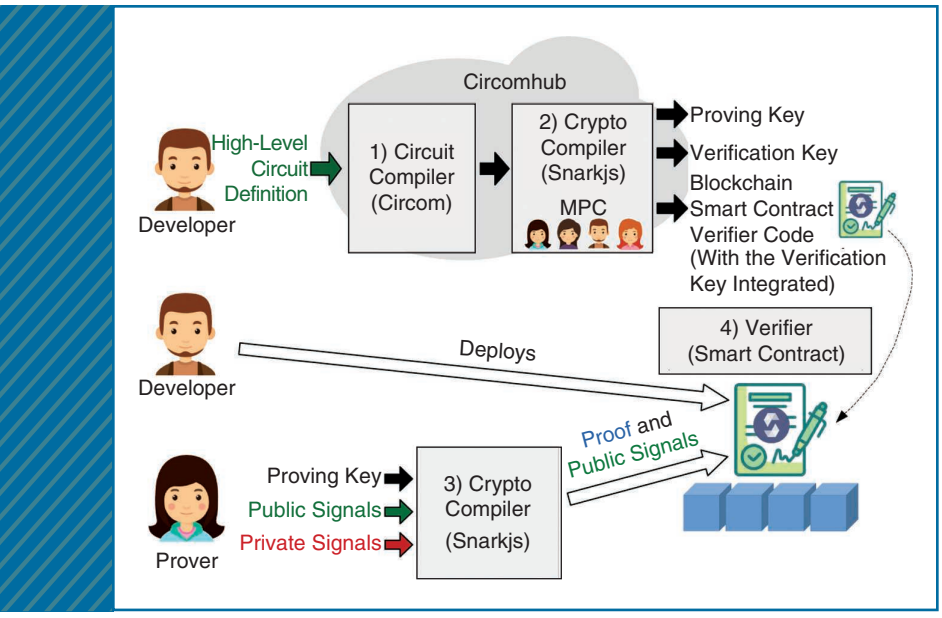

FIGURE 4. Our framework tools.

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. Therefore, if the prover would sign a transaction, he/she would lose his/her anonymity. A simple solution to obfuscating a member's identity 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, which is a decentralized network of relayers.[14]

Developer-friendly languages like circom, which 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 become familiar with our tools and explore the possibilities they offer.

Acknowledgments

This research is supported by the Ethereum Foundation Ecosystem Support, privacy and security in blockchain (RISEBLOCK, Id: PID2019-110224RB-I00), intelligent, interoperable, integrative and deployable open source MARKETplace with trusted and secure software tools for incentivizing the industry data economy (i3-MARKET, European Commission, H2020, Id: 871754), a software architecture for rate-control over integrated satellite-terrestrial networks (ARPASAT, Id: TEC2015-70197-R), Information Security Group (ISG, Id: 2014-SGR-1504), Photoreal REaltime Sentient ENTity (PRESENT, European Commission, H2020, Id: 856879), and Técnicas avanzadas de cadenas de bloques para la internet de las cosas (Id: RTI2018-102112-B-I00).

References

1. R. G. Durán, D. Yarlequé-Ruesta, M. Bellés-Muñoz, A. Jimenez-Viguer, and J. L. Muñoz-Tapia, "An architecture for easy onboarding and key life-cycle management in blockchain applications," *IEEE Access*, vol. 8, pp. 115,005–115,016, Jun. 2020. doi: 10.1109/ACCESS.2020.3003995.
2. V. J. Morkunas, J. Paschen, and E. Boon, "How blockchain technologies impact your business model," *Bus. Horiz.*, vol. 62, no. 3, pp. 295–306, 2019. doi: 10.1016/j.bushor.2019.01.009.
3. Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. 2017 IEEE Int. Congr. Big Data (BigData Congr.)*, Honolulu, HI, pp. 557–564. doi: 10.1109/BigDataCongress.2017.85.
4. J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016*, vol 9666, M. Fischlin and J. S. Coron, Eds. Berlin: Springer-Verlag, 2016, pp. 305–326.
5. S. Bowe, A. Gabizon, and I. Miers, "Scalable multi-party computation for zk-SNARK parameters in the random beacon model," *IACR Cryptology ePrint Archive*, Bellevue, WA, Rep. 2017/1050, 2017.
6. E. Ben-Sasson *et al.*, "Zerocash: Decentralized anonymous payments from bitcoin," in *Proc. IEEE Symp. on Security and Privacy*, San Jose, CA, 2014, pp. 459–474. doi: 10.1109/SP.2014.36.
7. D. Khovratovich and M. Vladimirov, "Tornado privacy solution: Cryptographic review. Version 1.1," ABDK

ABOUT THE AUTHORS

MARTA BELLÉS-MUÑOZ is a Ph.D. student at Pompeu Fabra University, Barcelona, 08018, Spain, with the Department of Information and Communications Technology, in collaboration with iden3, Zug, CH-6300, Switzerland. Her research interests include security and efficiency of arithmetic circuits for zero-knowledge protocols. Bellés-Muñoz received her B.S. in mathematics from the Autonomous University of Barcelona and continued her master's studies at Aarhus University, where she focused on the study of elliptic curves and isogeny-based cryptography. Contact her at marta.belles@upf.edu.

JORDI BAYLINA is the cofounder of iden3, Zug, CH-6300, Switzerland, and the main contributor to circom and snarkjs. He is also a part of the White Hat Group. Baylina received his B.S. in telecommunications engineering from the Polytechnic University of Catalonia. His current research interests include applied cryptography, zero-knowledge proofs, and distributed ledger technologies. Contact him at jordi@baylina.cat.

VANESA DAZA is an associate professor at Pompeu Fabra University, Barcelona, 08018, Spain, with the Department of Information and Communications Technology. Her main research interests are distributed cryptographic techniques to improve security and privacy of blockchains. Daza received her Ph.D. in mathematics from the Polytechnic University of Catalonia. She is an associate editor of *IEEE Transactions on Dependable and Secure Computing* and *IEEE Transactions on Information Forensics and Security*. Contact her at vanesa.daza@upf.edu.

JOSÉ L. MUÑOZ-TAPIA is an associate professor and director of the Master's in Blockchain Technologies at the Polytechnic University of Catalonia, Barcelona, 08034, Spain, with the Department of Network Engineering. His research interests include applied cryptography and distributed ledgers. Muñoz-Tapia received his Ph.D. in security engineering. Contact him at jose.luis.munoz@upc.edu.

Consulting, Tallinn, Estonia, 2019. [Online]. Available: https://tornado.cash/audits/TornadoCash_cryptographic_review_ABDK.pdf

8. "Hermez white paper," Hermez Network, Zug, Switzerland, White Paper, Oct. 2020. Accessed: Dec. 29, 2020. [Online]. Available: https://hermez.io/hermez-whitepaper.pdf
9. "Circom featured project," Ethereum Foundation Ecosystem Support, 2020. https://esp.ethereum.foundation/en/projects/circom/ (accessed Dec. 29, 2020).
10. "Circom: Circuit compiler for zero-knowledge proofs," GitHub, 2020. https://github.com/iden3/circom (accessed Dec. 29, 2020).
11. "snarkjs: JavaScript Implementation of zk-SNARKs," GitHub, 2020. https://github.com/iden3/snarkjs (accessed Dec. 29, 2020).
12. "Circomlib: Library of Circom templates," GitHub, 2020. https://github.com/iden3/circomlib (accessed Dec. 29, 2020).
13. S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (EdDSA)," IRTF, RFC: 8032, 2017.
14. "Ethereum gas station network." GSN. https://www.opengsn.org/ (accessed Dec. 29, 2020).