ZKPs for Privacy-Preserving Smart Contracts and Transactions

Ravital Solomon[1]

[1]NuCypher
ravital@nucypher.com

Abstract

Zero-knowledge proofs are often a necessary component in conducting private transactions on a public, distributed ledger. There are many options when choosing such an efficient zero-knowledge proof yet understanding all the tradeoffs between them can be a difficult task.

In this paper, we investigate 3 major efficient zero-knowledge proof systems (i.e. SNARKs, STARKs, Bulletproofs) along with their tradeoffs. We then go on to explore their applications to private transaction and smart contract schemes. Finally, we offer some recommendations and thoughts for NuCypher's direction in this space.

Contents

1 Abbreviations

CRS. Common Reference String
DHP. Diffie-Hellman Problem
DLP. Discrete Log Problem
DPC. Decentralized Private Computation
ECDSA. Elliptic Curve Digital Signature Algorithm
EIP. Ethereum Improvement Proposal
FHE. Fully Homomorphic Encryption
KoE. Knowledge of Exponent
LBC. Lattice-Based Cryptography
LWE. Learning with Errors
MPC. Multiparty Computation
NIZK. Non-Interactive Zero-knowledge
SHE. Somewhat Homomorphic Encryption
SIS. Short Integer Solution
UTXO. Unspent Transaction Output
ZK. Zero Knowledge
ZKP. Zero-Knowledge Proof

2 Introduction

We explore different zero-knowledge proof protocols to determine which protocol is the most promising for privacy-preserving smart contracts.

We begin by looking at the most popular ZKP schemes—namely SNARKs, STARKs, and Bulletproofs. To this end, we explore the use of trusted setups, different hardness assumptions, and recent projects in the private trans-

action/smart contract space. Finally, we provide some of our recommendations and consider future directions for research at NuCypher.

Note: We do not make the distinction between zero knowledge proofs vs. zero knowledge arguments. When we say a transaction is "confidential," we're referring to a transaction in which the amount being transferred is hidden. When we say a transaction is "anonymous," we're referring to a transaction that is both confidential and hides the sender/receiver. When we use the term zk-SNARKs, we are referring to the non-universal, QAP-based SNARKs [4].

Timeline

April 2019: Document completed.

April 2020: This document was originally written for internal usage as a preliminary study on efficient zero-knowledge proofs. Our goal was to explore different zero-knowledge proof protocols to find the most appropriate one to use in a potential private smart contract scheme (though what properties such a scheme would provide was not yet decided).**We completed this document in April 2019 so it no longer reflects recent advancements in the space.** ZKP research moves quickly and there are a number of new, interesting papers—PLONK [17], SNARKs from Dark Compilers [12], and Fractal [14] to name a few—that are not discussed here.

3 Zero-Knowledge Proofs

There are many good resources for understanding the basics of zero-knowledge proofs. We will provide a brief, non-technical introduction to the concepts for unfamiliar readers.

Two parties—namely, a **Prover** and a **Verifier**—participate in a ZKP. The Prover wants to convince the Verifier of some fact. However, the Prover does *not* want the Verifier to learn anything from this interaction (other than that the fact is true). Specifically, the Prover does not want the Verifier to learn information that might help the Verifier to prove the same fact to someone else.

A ZKP should satisfy the following 3 properties:

1. *Soundness:* If that statement is *false*, the Prover will *not* be able to convince the Verifier that the statement is true.

2. *Completeness:* If the statement is true and both parties are behaving honestly, the Prover will convince the Verifier that the statement is true.

3. *Zero Knowledge:* If the statement is true, the Verifier will *only* have learned that the statement is true at the end of the protocol.

SNARKs, STARKs, and Bulletproofs are popular types of zero-knowledge proof systems. Compared to most ZKPs, they're very efficient in terms of

(proof) size, and time—which is why they're used in the distributed computing setting. To prove properties about encrypted data (whether it be for private transactions or more complex private smart contracts), we will need a ZKP.

4 Privacy-Preserving Smart Contracts

We are interested in investigating the tools we need to build "privacy-preserving smart contracts." PPSCs can be viewed as an extension to private transactions as they usually include some of the same operations (e.g. auctions). In a private transaction, we would like both the amount being sent and the balances of the users' to be "hidden" at the minimum. We refer to these types of transactions (where the inputs/outputs are hidden) as "confidential transactions." We can also ask that the identities of the users involved be hidden but this a stronger condition (we refer to these as "anonymous transactions").

Since the transfer amount and balances are hidden in a confidential transaction, we need some way to verify that the operations were performed correctly. This is where a ZKP is needed; it allows the sender to prove to others that their transaction is "valid" and performed correctly. Some popular choices of ZKPs in the private transaction setting include zk-SNARKs (as seen in Zcash), STARKs, and Bulletproofs (as seen in Monero). These proof systems use different types of setup processes, are based on different hardness assumptions, and offer various tradeoffs.

Since ZKPs are an integral part of a confidential transaction/PPSC scheme, we will further investigate the different ZKPs to understand which one may be the most useful for us.

5 Trusted Setups: Common Reference String Model

In a "trusted setup," one or more parties come together to generate a "common reference string" that will be used as public parameters in the ZKP protocol. The CRS should be independent of any statement we'll prove. Usually, an MPC protocol takes place to generate the CRS. When such a process takes place, the parameters are "secure" so long as at least one party was "honest" (i.e. destroyed his secret randomness used to generate the CRS). An example of a well-known private transaction scheme that uses a trusted setup is Zcash.

One of the biggest benefits of the CRS model is the efficiency gained by storing expensive pre-computations in the public parameters. This can help to significantly cut down on the proof size. When we require very small proof sizes, working in the CRS model may be the best option available.

However, the CRS model requires some degree of trust in the parties that generated the common reference string. If the setup process is compromised (e.g. all the parties are "dishonest" and collude together by keeping the secret randomness used to generate the public parameters), then it may be possible to

violate the "soundness" property of the ZKP. One example of an issue arising from this is in Zcash where (if the setup were compromised) it's possible to create tokens out of thin air [23].

An interesting improvement to the CRS model is the "updateable" CRS which allows for a notion of "updateable trust" [21]. The above CRS model (as proposed) is static. That is to say we assume the trusted setup process takes place once at the start of the protocol with a fixed number of parties. An updateable CRS instead allows new parties to continue joining the "setup" process and contribute to the creation of the public parameters at any point in time [21]. With an updateable CRS, the parameters are "secure" after at least one "honest" party (who destroyed his secret randomness) joins. That means that even if all the previous parties in the setup were dishonest/cheating, once an honest party joins the setup the parameters from that point on are "secure." This allows for less trust than in the standard CRS with trusted setup. Unfortunately, such an updateable CRS introduces a large degree of inefficiency into our protocol since we'd have to "keep track" of the updated public parameters and the corresponding proofs used to update them.

6 Hardness Assumptions: Discrete Logs vs. Lattices

We can consider proof systems that rely on the hardness of the discrete log problem (along with Diffie-Hellman variants) or on lattice assumptions (LWE, SIS, etc.).

It's quite rare in practice to see proof systems using lattice-based cryptography (LBC) as they're much less efficient than proof systems based on discrete logs/DHP. Proving general linear relations of the form $\mathbf{A}\vec{x} = \vec{b}$ is generally very inefficient with proof sizes even in the MBs (see [20] for details). LBC does offer quantum resistance which may be of importance depending on the specific use case.

On the other hand, proof systems based on the discrete log problem and its variants are very common in the cryptocurrency space. While they do not offer quantum resistance, they're arguably easier to work with, better understood, and the proofs significantly cheaper. We will see them used in Bulletproofs (and correspondingly in Zether and Quisquis) [10, 9, 16].

Note: You may wonder at this point "under which category do zk-SNARKs fall?" They use the "Knowledge of Exponent" assumption which is arguably a much stronger cryptographic hardness assumption than DLP/DHP. If interested, see Vitalik Buterin's "Zk-SNARKs: Under the Hood" series for an accessible introduction to the topic [11].

7 Zk-SNARKs and Recent Improvements

Zk-SNARKs (**Z**ero-**k**nowledge **S**uccinct **N**on-interactive **Ar**guments of **K**nowledge) gained fame when they were used by Zcash to perform anonymous transactions (i.e. transactions in which the sender, receiver, amount, and their balances are hidden) [23, 18]. They require a trusted setup, are based on strong cryptographic assumptions (KoE), but offer no post-quantum security. However, they enjoy very small proof sizes (in part thanks to the trusted setup), and fast verification time. Since a new CRS has to be generated for each new circuit, zk-SNARKs are best suited to use cases where we're proving the same statement many times (see **SONIC** below) [21].

7.1 SONIC: Zk-SNARKs from Linear-sized Universal and Updateable Structured Reference Strings

SONIC, presented at CRYPTO '18, improves upon zk-SNARKs by introducing the notion of a "universal" and "updateable" linear-sized CRS. SONIC allows for an "updatable" trust model (see discussion in **Trusted Setups** section above) which is weaker than the trust normally needed for zk-SNARKs [21]. This places SONIC somewhere in between trusted and untrusted setup. With the updateability comes "universality" which allows for a single reference string to be used to prove any relation of some bounded size [21]. This feature makes SONIC more flexible in its use case.

In SONIC, there's a "global" CRS and a "derived" CRS [21]. The updateable "global" CRS is not linear-sized—it's quadratic—whereas the "derived" CRS is linear-sized. Using SONIC, we would still need to store the two most recent global CRS and all the update proofs between them (and in order) [21]. This obviously limits the actual "updateability" of the CRS in practice. Another point worth noting is the unusual security model in which this is proven—the algebraic group model: "it is so far unknown how the algebraic group model relates to the KoE assumptions, which are used to build every known SNARK that has been proven secure in the standard model" [21].

8 Zk-STARKs

Zk-STARKs (**Z**ero-**k**nowledge **S**calable **T**ransparent **Ar**guments of **K**nowledge) were formally introduced in late 2017/early 2018 to address some of the problems of zk-SNARKs [7]. Namely, zk-STARKs provide a proof system that does not require a trusted setup (or even a reference string!). They also provide quantum resistance (since STARKs use hash-based cryptography), and are supposedly more scalable than SNARKs [7]. However, STARKs are very much an active work in progress with research being led by the company Starkware. Starkware is currently focused on further improving the scalability of STARKs [1].

Unfortunately, STARKs (in their present state) do not appear to be very efficient in practice. The biggest issue is proof size—zk-STARKs are 1-2 orders of

magnitude larger than Bulletproofs and 2-3 orders of magnitude larger than zk-SNARKs [3]. However, they do offer fairly competitive proving and verification times for transactions compared to zk-SNARKs. While they do appear "scalable" in terms of algorithmic complexity, they offer worse asymptotic runtimes than either SNARKs or Bulletproofs (see Table 1 in [21]). How zk-STARKs perform with an appropriate number of bits of security is also unclear; in the paper introducing STARKs, the authors only consider a zk-STARK system providing 60 bits of security (see section 1.3.2 in [7]).

We would describe STARKs as a very recent and untested piece of technology in the sense that they have yet to be deployed in any novel application (though Starkware will deploy "StarkDEX," a "scalability engine" for decentralized exchanges in April 2019; see Starkware's Medium blog for further details) [1].

9 Bulletproofs and Recent Improvements

Bulletproofs are a circuit proof system that are best suited for "range" proofs [10]. Unlike zk-SNARKs, they do not have a trusted setup but do require a reference string that the Verifier needs to access. It's worth noting that this reference string can be generated "on the fly" but the process can be quite costly. As one can imagine, the lack of trusted setup translates into larger proofs and worse proving/verification times. Although zk-SNARKs offer very short proving/verification times which make them immediately appear superior, we have to keep in mind the overhead involved in storing a huge CRS (which could easily be in the GB) and the limited proving capabilities zk-SNARKs offer since we need to generate a new CRS for a new circuit. Finally, Bulletproofs are based on a fairly weak and widely understood hardness assumption (DLP) making it superior to zk-SNARKs in that regard.

A number of interesting projects/works have been recently released using Bulletproofs—namely Zether, Quisquis, and ZkVM [9, 16, 5]. We will also see variants of Bulletproofs being used to prove properties of ciphertexts and plaintexts for FHE (see **On Integrating FHE** below for further details) [15].

Note: If interested in concrete numbers and comparisons with zk-SNARKs, see [3].

Sigma-Bullets

Sigma-Bullets (also referred to as Σ - Bullets) were recently introduced in Zether with the goal of making Bulletproofs "interoperable" with Sigma protocols [9]. Like Bulletproofs, Sigma-Bullets are "trapdoor free" and have log-sized proofs. Recall that Bulletproofs are best suited for proving arithmetic statements. Sigma-Bullets will instead allow us to prove statements on "algebraically encoded data" (see "Sigma-Bullets" section 3.3 and G in [9] for further details).

In Zether, they're ultimately used to combine range proofs (from Bulletproofs) with ElGamal encryptions (that are used to identify users and store messages).

10 Recent Projects in the Space

Below, we provide short overviews of a few promising projects that are looking to build privacy-preserving transactions and/or smart contracts.

Zether is arguably the most promising work for us—with clear security proofs and protection against various attack scenarios [9]. They focus on building privacy-preserving smart contracts using an account-based model. While their work is compatible with Ethereum, it's not very practical given Ethereum's current limitations (namely how expensive certain elliptic curve operations are). They make use of "Sigma-Bullets."

Quisquis presents a new cryptocurrency providing anonymity [16]. While this route (creating an entirely new cryptocurrency to build privacy-preserving smart contracts) is not preferable, they introduce some novel concepts—namely they combine the UTXO and account models together and use updateable public keys. They fail to address front-running (public key can be updated before transaction is processed thus rendering the transaction invalid) and it's not currently clear if privacy-preserving smart contracts can be built with their work [9].

AZTEC addresses how to build efficient confidential transactions that are compatible with Ethereum [22]. Little information is provided on how to implement an anonymous transaction or how to build smart contracts that go beyond atomic swaps. AZTEC uses a UTXO model which seems a bit unusual in combination with Ethereum's account model. However, AZTEC is significantly more efficient than Zether when performing a confidential transaction on Ethereum (their whitepaper quotes that an AZTEC proof costs 840,000 gas to verify) [22] . Unfortunately, to achieve this efficiency, AZTEC requires a trusted setup/CRS.

Zexe introduces a cryptographic primitive they refer to as a "decentralized private computation" scheme [8]. As it comes from (some of) the same authors as Zcash, it serves as no surprise that their work builds upon Zcash. Using a DPC scheme, users can perform computations offline and then create transactions that provide ZKPs (via zk-SNARKs) attesting to the correctness of these computations. This allows for some notion of "function privacy." Unfortunately, their work seems to require the use of numerous trusted setups and very expensive proofs.

10.1 HAWK: Private Smart Contracts

Note: We include only a very brief discussion on HAWK as the work is not particularly relevant to our goals/requirements for a PPSC scheme. However, we feel it's important to include in the document as it deals with the creation of private smart contracts.

HAWK uses zk-SNARKs and suffers from a number of weaknesses that don't make it particularly useful for our privacy-preserving smart contract setting [19]. While HAWK does provide privacy, it requires arguably *even more trus*t than the usual trusted setup for zk-SNARKs. In HAWK, users must entrust their private inputs to some designated "manager" who'll generate the zk-SNARK (see "Related Work" in Zether paper) [19, 9]. As with zk-SNARKs, the CRS is circuit-dependent implying that we'd need to somehow generate a new CRS for each new smart contract (the circuit dependency also calls into question whether it can be integrated in the future with FHE successfully). Another (potential) issue is that the number of participants is fixed—thus limiting the types of smart contracts we can consider.

However, the advancements in SONIC (particularly the universality) may allow for some of the weaknesses in HAWK to be ameliorated [21, 19].

10.2 Zether

Zether introduces privacy-preserving smart contracts that can be built on top of Ethereum that do not require a trusted setup [9]. Namely, they introduce a way to send Ether "confidentially" (i.e. without revealing the transaction amount) and then also "anonymously" (i.e. without revealing either the sender or receiver). They address front-running and provide protection against replay attacks along with overdraft safety. Unlike AZTEC or Quisquis, Zether's paper details how to use their work to build privacy-preserving smart contracts. Since Zether uses an account-based model and also keeps track of "state," it's more apparent to the reader how Zether may support smart contracts than AZTEC or Quisquis.

As part of their building blocks, they use the ElGamal encryption scheme, Sigma-Bullets, and Schnorr signatures. They use the ElGamal encryption scheme since it's an additively homomorphic encryption scheme that will allow them to store account balances and other messages in the exponent. Sigma-Bullets will allow Bulletproofs to be interoperable with Sigma protocols (necessary for combining range proofs with ElGamal encryptions) [9].

Unfortunately, Zether is not very practical when implemented on Ethereum currently. According to the paper's estimates, a confidential transaction costs approximately 7.2 million gas (with almost 90% of the cost coming from elliptic curve operations) [9]. If EIP 1108 were implemented, they note that the cost of a confidential transaction would be only $\sim$2 million gas [9, 13]. While Zether provides a framework for creating anonymous transactions based on ring signatures, an anonymous transaction's cost exceeds Ethereum's gas limit [9]. Only one anonymous transaction can be performed per each time period (referred to as "epochs" in the paper) [9]. Additionally, the authors make some assumptions with regard to the network's synchronization [9].

Despite Zether's practical limitations, their work is valuable when it comes to exploring how to build privacy-preserving smart contracts. The use of an account-based model also makes Zether well suited for the smart contract setting. The main drawback to this work is its current feasibility on Ethereum.

10.3 Quisquis

Quisquis is a new anonymous cryptocurrency that requires no trusted setup [16]. It was initially designed to improve upon the scalability and privacy guarantees of both Zcash and Monero [16, 23, 2]. It uses updateable public keys (to define the notion of an "updateable" account), additively homomorphic commitment schemes that are "key anonymous," and zero-knowledge proofs based on DHP/DLP [16]. It combines the UTXO and account model together while using Bulletproofs to ensure certain conditions are satisfied. The proofs are significantly larger in size than those in Zcash but still manageable [16]. For reference, with an anonymity set size of 16, Quisquis states that transactions take around 471ms to compute, and that their proofs would be ∼13kB in size (for reference Zcash's proof size is approx. 30 bytes) [16].

Similar to Monero, it allows for "plausible deniability" so that users can deny having ever participated in a confidential/anonymous transaction [16, 2]. Taking inspiration from Bitcoin (but replacing UTXOs with the users' keys), in a Quisquis transaction, all of the input public keys in the UTXO set are replaced with new output public keys (with users being able to "destroy" accounts with balance 0). Thus, the authors refer to the public keys as being "updateable" since they are "updated" after each transaction. This allows Quisquis, unlike Zcash and Monero, to achieve non-monotonic UTXO set size growth [16].

Each account is represented by a public key (specifically an "updateable" public key), and a commitment (which contains the balance associated with that public key). Thus, we need the commitment scheme to be additively homomorphic (to allow for updating the balances) and key anonymous (to make it difficult to tell apart commitments belonging to different public keys). They use an updateable public key scheme based on the hardness of DHP since it will allow them to make use of DLP-based ZKPs.

While Quisquis provides security proofs to ensure anonymity and theft prevention, there is no clear solution to prevent "front-running" [16, 9]. By "front-running," we consider the situation in which Alice wants to send Eve some money and thus needs to generate a NIZK proof with respect to her current balance. Suppose Bob comes along and wants to send Alice some money and that his transaction is processed before Alice's. Now Alice's proof is no longer valid and thus her transaction will be rejected. This can be problematic if fees are involved (to add transactions to the blockchain). It is also unclear how to build privacy-preserving smart contracts using their model. In addition, there's some significant inefficiencies in updating balances if someone has received coins but does not know the amount ahead of time—in the worst case, they'd have to brute check every possible amount.

Quisquis serves as another example of the value of Bulletproofs in proving relations about anonymous transactions. There is a pattern emerging with regards to the properties we want our commitment/encryption scheme to satisfy—namely, it should be "homomorphic." For us, Quisquis does not provide the best path forward in creating privacy-preserving smart contracts given their complex UTXO-account hybrid model and vulnerability to front-running at-

tacks.

10.4 AZTEC

AZTEC (**A**nonymous **Z**ero-knowledge **T**ransactions with **E**fficient **C**ommunication) details how to perform confidential transactions efficiently on Ethereum [22]. They use a UTXO-based model and require a trusted setup to achieve (relative) efficiency. The CRS (which is only needed by the Prover to construct ZKPs) contains elliptic curve points and scales linearly with the size of the range (i.e. the upper bound on the value of all notes in the system) used in their range proof. For reference, in their implementation, the size of their range is $2^{25} - 1$ [22].

Each AZTEC note comprises of a tuple of elliptic curve commitments that contain the balance, along with the viewing and spending keys [22]. As part of their building blocks, they use commitment schemes whose hardness is based on the DHP, Sigma protocols, and ECDSA signatures (carried over from Ethereum so that an Ethereum private key can serve as an AZTEC note spending key).

According to their estimates, it costs ∼800-900k gas to verify their zero-knowledge proof [22]. Their use of a UTXO-based model makes it unclear as to how one might perform computations that require knowledge of the current state. They mention in their whitepaper that "an AZTEC verifier must be able to validate the existence of unspent AZTEC notes (and therefore have access to some form of persistent state), but the manner in which this is achieved is beyond the scope of this paper" [22].

In addition, there are potentially a host of security issues that may arise from their combination of Ethereum's account-based model with AZTEC's UTXO-based model. They do not provide any analysis to show the reader that this hybrid model does not introduce new security issues. Many details are omitted—including how to perform anonymous transactions. It's worth noting that this is another paper that uses DHP/DLP and Sigma protocols to achieve confidential transactions. It may also be worth considering using a CRS as AZTEC does if we want to achieve better efficiency in our privacy-preserving smart contracts.

10.5 Zexe

Zexe (**Z**ero Knowledge **Exe**cution) is a ledger-based system that allows users to perform offline computations and then provide publicly-verifiable transactions attesting to the correctness of these offline computation (via zk-SNARKs) [8]. In Zexe, the authors further define the notion of a "decentralized private computation (DPC)" scheme which is a generalization of the above. They build upon Zcash's work by allowing not only for "input/output privacy," but also for "function privacy." Their construction uses a UTXO-like model since each transaction in their scheme consumes old records and creates new ones [8]. The building blocks include a commitment scheme, zk-SNARKs, and a randomizable signature scheme (to prevent the "linking" of signatures in the delegable

setting).

Transaction sizes depend linearly on the input and output records ((32 × |input records|)+(32×|output records|)+820 bytes) [8]. Verification is fast—requiring ≤ 50 milliseconds [8]. The tradeoff is the transaction generation time—approximately a minute for just 2 input and output records [8]. This calculation is for a machine with an Intel Xeon 6136 CPU at 3.0 GHz with 252 GB of RAM with 12 cores/threads [8].

They require a trusted setup to obtain the public parameters of the scheme (which allow them to create/verify transactions), and subsequent trusted setups for each new user-defined application. Specifically, each record contains birth/death predicates specified by the user and each of these predicates require their own parameter generation process. For more details, please see Sections 2.4, 2.6, and 3.1 of their paper [8].

As the zk-SNARKs can be quite expensive to generate, their scheme allows users to "outsource" expensive proof computations to "untrusted" workers ("delegable DPC") [8]. However, this requires that the user entrust secrets about the transaction to the worker (so that the worker can generate the cryptographic proof). The user meanwhile retains a separate key that will allow him to authorize this transaction. The "untrusted" worker learns information about the user and his transaction in the process of generating this proof (the authors of the paper leave this problem for future work). If interested further, please see Section 5 of the paper for more info [8]. Considering the machine requirements needed for generating the zk-SNARKs efficiently in the paper (256GB of RAM to name one), this likely means that most users will have to delegate their cryptographic proofs to untrusted workers (calling into question the privacy that's actually achievable in a real-world setting) [8].

As it currently stands, Zexe does not support "stateful" computations that require knowledge of a record's ledger position (which would be necessary to implement a "time lock" for example). However, they argue that it's possible to incorporate a transaction's position on the ledger to achieve such applications (see Appendix C in their paper for details) [8].

11 On Integrating FHE

Suppose a number of users are performing some computations on encrypted data (i.e. fully homomorphic encryption). Our primary concern here is that the ciphertexts are in fact "valid," well-formed ciphertexts. We may also want to know if the corresponding plaintexts satisfy certain relations. To solve these problems, we can require the users to provide ZKPs attesting to these facts.

However, the immediate question we must ask ourselves is "what sort of ZKPs should we use?" FHE uses LBC yet the most efficient zero-knowledge proof schemes do not use LBC. It's worth noting that to prove even simple relations of the form $\mathbf{A}\vec{x} = \vec{b}$ (matrix-vector equation), we end up with proofs that are in the MB size range (see "Lattice-Based Zero Knowledge Arguments for Linear Relations" and "Short Discrete Log Proofs for FHE and Ring-LWE

Ciphertexts" for more details) [20, 15].

In "Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts," we find that we can efficiently prove properties about FHE ciphertexts/plaintexts by using discrete log based proofs—namely a variant of Bulletproofs [15]. The main idea behind their technique involves:

- noticing that a FHE ciphertext can be represented by the matrix-vector equation $\mathbf{A}\vec{x} = \vec{b}$ (where $\mathbf{A}$ is the public key, $\vec{x}$ is the plaintext message and randomness, and $\vec{b}$ is the ciphertext)
- committing to the coefficients (we're working over a polynomial ring) of vector $\vec{x}$ using Pedersen commitments
- finally, using "efficient" DL-based ZKPs (i.e. Bulletproofs) to prove that $\vec{x}$ satisfies the above equation and/or prove relations involving the plaintext in $\vec{x}$.

They estimate that the size of such a proof showing validity of a Ring-LWE ciphertext would be approx. 1.3kb [15].

12 Recommendations

We begin by outlining the conditions we'd like our protocol to satisfy and which protocols fulfill these conditions. We'll then offer our recommendations at the end of this section.

Necessities

1. *Is the protocol based on weak, well-understood hardness assumptions?*
   We have to exclude zk-SNARKs if we require our protocol to be based on well-understood, weak cryptographic hardness assumptions [11]. The "Knowledge of Exponent" assumption used in zk-SNARKs is arguably a strong hardness assumption. This correspondingly excludes consideration of SONIC since they use the algebraic group model; it's also not yet understood how KoE relates to the algebraic group model [21].

2. *Can we build additively homomorphic commitment or encryption schemes using this type of cryptography?*
   Cryptography based on the hardness of DLP/DHP and LBC both satisfy this condition. DL-based cryptography is arguably easier to work with and we have "practical" proof schemes readily available.

3. *Will our ZKP protocol produce proofs of an "acceptable" size? (i.e. <<10kb)*
   This condition leads us to excluding zk-STARKs as they currently produce proofs between 40-200kb in size [3]. Recall that Bulletproofs have proofs of <1.5kb in size, whereas zk-SNARKs have proofs of <300bytes in size (see [3]).

4. *Does our protocol require a trusted setup?*
   The reader may ask at this point: "what's wrong with using a trusted setup?" We would argue that in the smart contract setting it may be difficult to find enough parties to contribute to the trusted setup process (important so that other users can reasonably believe that the trusted setup hasn't been compromised). If only 1-2 parties contribute to the CRS, it would likely lead to a lack of trust in the setup process and suspicion that the contributing parties have kept the trapdoor for their own gain. It also seems a little cumbersome to generate a new CRS for each new contract (assuming the CRS is non-universal). Thus zk-SNARKS should be excluded from consideration.

5. *Can we use our ZKP protocol to prove properties of FHE ciphertexts/plaintexts?*
   As we saw earlier in the section **On Integrating FHE**, Bulletproofs have recently been used to prove properties about FHE ciphertexts [15]. Thus, they appear to be the clear winner in this category. Zk-STARKs may potentially be an option but it's very early to say and would require quite a bit of R&D. Zk-SNARKs do not appear to be a good choice given their circuit-dependent CRS.

Nice to Have

1. *Do we use cryptography that's post-quantum?*
   Only zk-STARKs are able to satisfy this condition (when considering zk-SNARKs, Bulletproofs, and zk-STARKs). A lattice-based proof system would satisfy this condition but we do not have any efficient proof schemes yet.

2. *Do we have very fast proving/verification times? (i.e. proving times of <5 seconds, verifying times of <100ms)*
   Zk-SNARKs and zk-STARKs satisfy this condition [3]. Bulletproofs are quite a bit slower but can potentially be batched for better efficiency (to at least improve upon verification times) [10]. Specifically, Zether suggests an optimization in which a service provider combines many transactions together to form a single transaction and finally sends it to the Zether contract [9]. However, if a single transaction is invalid in the batch, then the verification process will fail. For further details, see Section 7.3 in [9].

3. *Has it been deployed or used in novel applications prior?*
   Zk-SNARKs and Bulletproofs have been used in a number of novel applications as we've seen throughout this paper [23, 9, 2]. Zk-STARKs are a very recent innovation and have yet to be deployed in any novel application.

We believe that Bulletproofs, despite their limitations, are the best proof system to currently use for building privacy-preserving smart contracts. More generally, crypto-systems and schemes based on the hardness of DLP/DHP appear to offer more immediate promise since they're highly efficient and relatively

cleaner to work with. Bulletproofs have already been successfully used to deploy privacy-preserving smart contracts (e.g. Zether) and prove properties of FHE ciphertexts ("Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts") [9, 15]. Thus, it seems like the basic building blocks are already available to achieve our goals.

13 A Potential Product

We provide further insight into why we've chosen Bulletproofs as our ZKP protocol and our vision of a "compatible" product suite. This requires us to first define two products we envision creating:

1. FHE + Proof of Ciphertext Validity
2. Privacy-Preserving Smart Contracts

FHE + Proof of Ciphertext Validity

We'll begin by providing a motivating example. Suppose we have a group of users who want to perform some FHE computation using the same public key (on a distributed ledger). One such example might entail company employees encrypting their monthly expenses and adding it to the running total. Employee X, who is in charge of company expenses, holds the private key that will allow him to decrypt the final value to determine the total monthly expenses of all employees. The concern we might have here is "have the users provided valid ciphertexts?" We can also require the corresponding plaintexts provided by the employees to satisfy some public relation as specified by Employee X.

We can adapt the ZKP protocol from "Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts" to solve the above problem [15]. Recall that this entails using a variant of Bulletproofs to show ciphertext validity. Please note that we have not entirely solved the verifiable computation problem here. Rather, we have focused on solving problems that may arise from multiple users using FHE on a distributed ledger. However, we can also extend the solution to support the multi-key FHE setting.

Privacy-Preserving Smart Contracts

We want to design a smart contract scheme that provides "privacy preserving" features. In particular, we'd like for it to be possible for users to perform fully anonymous transactions (i.e. transactions that do not reveal the identities of the users or the transaction amount). It should also be possible to create confidential applications such as sealed-bid auctions and voting mechanisms from our work.

To solve the above, we will build upon the ideas presented in Zether [9]. Recall that Zether allows us to perform both confidential and anonymous transactions. We can also "lock" the Zether contract to other smart contract accounts to add confidentiality to new applications [9].

Combining the Two

Ideally, the two ideas described above are "compatible" in the sense that the proof protocols are "interoperable." This leads us to wanting to choose the same proof protocol for both products. As it currently stands, we have short efficient proofs of FHE ciphertext validity using a modified version of Bulletproofs and privacy-preserving smart contracts that allow for confidential/anonymous transactions via a (different) modification of Bulletproofs [15, 9].

Thus, we propose building upon ideas from both "Zether: Towards Privacy in a Smart Contract World" and "Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts." We believe that this may allow us to achieve the following goals:

- A "trustless" environment (since no trusted setup is needed with Bulletproofs)
- FHE on a distributed ledger with ciphertext validity (which can possibly be extended to the multi-key FHE setting) [15]
- Flexible smart contract design (no trusted setup, universal setting)
- Support of stateful computations (since we build upon Zether's account model)
- Compatibility with Ethereum (though not necessary!)
- Confidential and (potentially) anonymous transactions
- Ability to "add" privacy to other applications (via a Zether-like locking mechanism) [9]

We would like to emphasize that it's by no means necessary to use the same proof protocol to achieve both goals. However, by using the same protocol, we allow for the two products to (possibly) be more easily integrated together in the future.

14 Future Directions

Some future considerations include interoperability between different proof protocols, exploration of Sigma-Bullets and zk-STARKs, and exploration of lattice-based proof schemes in situations where proof size isn't of much concern.

Interoperability

The ZKP community has increasingly started to focus on the interoperability between different proof protocols [4]. For example—can we make our particular implementation of Bulletproofs "interoperable" with zk-SNARKs? This is an interesting potential avenue of research.

New Applications of Sigma-Bullets

As the goal of Sigma-Bullets is to make Bulletproofs more "interoperable" with Sigma protocols, we may wish to explore using Sigma protocols to prove lattice-based relations (instead of proving statements involving ElGamal encryptions).

Using Zk-STARKs

Zk-STARKs are a very recent development that have yet to be deployed in any novel application. However, they offer some impressive guarantees when it comes to proving/verification times and are also the only major proof system that's (potentially) post-quantum. It's worth keeping a close eye on Starkware to see if they can improve zk-STARKs and integrate them in an application. However, we do not imagine zk-STARKs as being a viable approach for privacy-preserving smart contracts for at least a year.

Lattice-based Cryptography

Lattice-based ZKPs are currently too inefficient to be of much practical use. Even state-of-the-art lattice-based proof protocols are not even remotely competitive with zk-STARKs and Bulletproofs (see [6] for cost table). As previously discussed, proofs of simple relations (in matrix-vector form) can result in proofs over 1MB in size. We may wish to explore use cases where this proof size is acceptable while keeping a further eye on improvements in this space as it's highly active.

Although a hybrid of lattice-based cryptography and DL-based cryptography, we'd recommend readers first start with "Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts," as the authors suggest that their techniques can potentially be applied to proving other relations in LBC [15]. As an example, they create a verifiable encryption/decryption scheme for Ring-LWE encryption in their paper [15].

For purely lattice-based ZKPs, we must first distinguish between wanting to prove statements involving arithmetic circuits vs. proving statements over the integers. For the former, we suggest starting with "Sub-Linear Lattice-Based Zero Knowledge Arguments for Arithmetic Circuits" (which was presented at CRYPTO '18) [6]. It's currently one of the most efficient protocols and provides sub-linear communication complexity. For the latter, we suggest starting with "Lattice-Based Zero Knowledge Arguments for Integer Relations" (also presented at CRYPTO '18) [20]. One of their contributions involves providing the first non-membership proof with logarithmic (in the set's cardinality) communication cost.

15 References

[1] Starkware. https://medium.com/@StarkWare.

[2] *Mastering Monero: The Future of Private Transactions*. Independently Published, 2018.

[3] zk-snarks vs. zk-starks vs. bulletproofs. Stack Exchange, 2018. https://ethereum.stackexchange.com/questions/59145/zk-snarks-vs-zk-starks-vs-bulletproofs-updated.

[4] Zkproof community reference, version 0.2. zkproof.org, December 2019.

[5] Zkvm. https://github.com/stellar/slingshot/tree/main/zkvm, February 2019. Stellar.

[6] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël Del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In *Annual International Cryptology Conference*, pages 669–699. Springer, 2018.

[7] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018:46, 2018.

[8] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 820–837, 2018.

[9] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. *IACR Cryptology ePrint Archive*, 2019:191, 2019.

[10] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[11] Vitalik Buterin. Zk-snarks under the hood. Medium, 2017. https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6.

[12] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. Cryptology ePrint Archive, Report 2019/1229, 2019. https://eprint.iacr.org/2019/1229.

[13] Antonio Salazar Cardozo and Zachary Williamson. Eip 1108: Reduce alt bn128 precompile gas costs. Ethereum Improvement Proposals, 2018. https://eips.ethereum.org/EIPS/eip-1108.

[14] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. Cryptology ePrint Archive, Report 2019/1076, 2019. https://eprint.iacr.org/2019/1076.

[15] Rafael del Pino, Vadim Lyubashevsky, and Gregor Seiler. Short discrete log proofs for fhe and ring-lwe ciphertexts. In *IACR International Workshop on Public Key Cryptography*, pages 344–373. Springer, 2019.

[16] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 649–678. Springer, 2019.

[17] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[18] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. Github. https://github.com/zcash/zips/blob/master/protocol/protocol.pdf.

[19] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[20] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Lattice-based zero-knowledge arguments for integer relations. In *Annual International Cryptology Conference*, pages 700–732. Springer, 2018.

[21] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2111–2128, 2019.

[22] Zachary Williamson. The aztec protocol. Available at https://github.com/AztecProtocol/AZTEC/blob/master/AZTEC.pdf, 2018.

[23] Zcash. What are zk-snarks? Zcash. https://z.cash/technology/zksnarks/.