The Future of Privacy-Preserving Smart Contracts

Ravital Solomon[1]

[1]NuCypher
ravital@nucypher.com

Abstract

Privacy-preserving smart contracts (PPSCs) are a burgeoning area of research as they are the natural extension of private transactions. We outline some design goals—efficiency, security, privacy, flexibility—for a PPSC scheme. We go on to emphasize the importance of flexibility and stress how Bulletproofs and fully homomorphic encryption can help us accomplish this goal. Next, we explore some possible lines of work for NuCypher in the PPSC space based on “Zether” and “Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts.” Finally, we provide a PoC of the verifiable encryption scheme from [13] and give some concrete estimates on the performance. The results appear to be promising and we advocate for pursuing a novel PPSC scheme using FHE and Bulletproofs.

Contents

1 Abbreviations

DLP. Discrete Log Problem
EC. Elliptic Curve
FHE. Fully Homomorphic Encryption
IBC. Inter-blockchain Communication
KEM. Key Encapsulation Mechanism
LBC. Lattice-based Cryptography
LWE. Learning with Errors
MPC. Multiparty Computation
NIZK. Non-interactive Zero Knowledge
PKE. Public Key Encryption
PPSC. Privacy-preserving Smart Contract
PQC. Post-quantum Cryptography
ROM. Random Oracle Model
SIMD. Single Instruction, Multiple Data
SP. Service Provider
TEE. Trusted Execution Environment
ZKP. Zero-knowledge Proof

2 Introduction

We provide what we hope will be the beginnings of a whitepaper on the development of a privacy-preserving smart contract scheme. More explicitly, this document will focus on the motivations and goals behind our scheme with less emphasis on the explicit technical details. A number of such private computation schemes have previously been proposed. We believe that ours will be different in that it optimizes for *flexibility* while still maintaining strong cryptographic security and privacy guarantees. To this end, we define 4 design goals of a PPSC system.

Our scheme is guided by two beliefs that we share with readers in the "Core Beliefs" section. We then introduce the cryptographic building blocks needed to accomplish our design goals. Finally, we provide a brief look into the system's architecture.

Timeline

October 2019: Document completed.

May 2020: This document was originally written to go alongside the talk given at Devcon ("The Future of Privacy-Preserving Smart Contracts"). Similar to the "ZKPs for Privacy-Preserving Smart Contracts and Transactions" document, this was written for internal usage. The content you find in the section "Possible Lines of Work" was originally part of a separate document titled "Using Bulletproofs" written in July 2019 for NuCypher. The "Implementation" section has been added May 2020.

3 Design Goals of PPSCs

A privacy-preserving smart contract (PPSC) should ideally satisfy the following 4 conditions:

- Efficient (taking physical resources into account)
- Secure (with regards to the **underlying security assumptions**)
- Private
- Flexible

We make a clear distinction between "secure" and "private" as they're different (although complementary) concepts. We will cover what each of these terms means to us below.

Very Important Note: ZKPs can also be discussed with respect to these 4 goals. *However,* the relationship between a *ZKP* satisfying a goal and a *PPSC scheme* satisfying the same goal is not as straightforward as may initially appear. One such example is Zether. The underlying ZKP—Bulletproofs—used is very "flexible." However, the entire scheme itself is **not** as flexible as only additive relations can be expressed in a private manner with the encryption scheme (since the encryption scheme is only additively homomorphic). Just because a underlying ZKP satisfies one of these goals, it is **not** true that the PPSC scheme will inherently satisfy the same goal. Other factors need to be taken into consideration (e.g. encryption scheme, accounts vs. UTXO model). **In contrast, if the underlying ZKP fails any of the above goals, it is practically always the case that the corresponding PPSC scheme will also.**

Efficient

Efficiency with respect towards PPSC scheme

By "efficiency," we're referring to how the system performs in practice. This can include time to perform transactions, time to verify transactions, setup time for the system, communication complexity, proof sizes, and potential for scalability (which can be aided through additional optimizations such as batching computations together). An additional (but crucial) consideration in achieving "efficiency" is the physical resources required to do so.

Efficiency with respect towards ZKP

It is misleading to claim any single ZKP system is "efficient" as there are many different aspects to efficiency. Of the 3 main ZKP systems, zk-SNARKs have the smallest proof sizes and the fastest verification times in practice [3]. However, the tradeoff is a costly setup process. Bulletproofs offer moderate proof sizes and no trusted setup process but are quite slow to verify unless "batched" together for better efficiency [10]. Zk-STARKs (asymptotically) offer fast proving and verification times but that comes at the cost of very large proof sizes and significant amounts of physical resources [3, 6].

Security Assumptions

Ideally, the security assumptions used in our scheme are based solely on well-understood cryptographic hardness assumptions such as the "Discrete Log Problem (DLP)" or "Learning With Errors (LWE)." Some recent proposals for PPSCs additionally rely on hardware (e.g. TEEs) or game-theoretic assumptions for security [11, 34].

We also take into account non-standard (cryptographic) hardness assumptions such as Knowledge-of-Exponent assumption (used by Zcash) and the Random Oracle Model (ROM) [33] . Trusted setups are also an important security consideration. Recall that if all parties in the trusted setup process collude, they can fabricate ZKPs and thus create coins out of thin air. Finally, we should consider what kind of adversary our scheme is secure against—a post-quantum one? A malicious one? A honest-but-curious one?

Private

By "privacy," we're referring to the *level* of privacy achieved by the scheme (confidentiality vs. anonymity) and *how* it's achieved. By "confidentiality," we refer to hiding both the inputs and outputs of the function. In a confidential transaction, the transaction amount and balances must be hidden. Anonymity includes hiding function inputs/outputs with the additional condition that the identities of the individuals involved must also be hidden. In an anonymous transaction, the transaction amount, users involved, and their respective balances are all hidden.

Confidentiality is often achieved through the use of additively homomorphic commitment or encryption schemes [32, 33, 9]. Of course, to *prove* that the operations were performed "correctly," and "honestly," a ZKP is needed. Anonymity can be more difficult to achieve and may involve both cryptographic and non-cryptographic techniques. These cryptographic techniques may include ring signatures, whereas non-cryptographic techniques may include the use of stealth addresses.

Flexible

By "flexibility," we're referring to how easily a particular scheme can be adapted to new settings.

We can further divide "flexibility" into two types—"**computation flexibility**," and "**setup flexibility**."

More specifically, we're interested in expressing any arithmetic circuit using our scheme so that we can represent a variety of functions in our smart contracts. We also would like to support "stateful programs." Schemes based on the UTXO model are also not very "flexible" as they don't easily support stateful computations in the same way an account-based system would. We group these types of concerns under "computation flexibility."

We are also interested in the **universal** NIZK setting—in which a single CRS supports proving any NP statement [4]. We would also like to avoid using trusted setups. We group these types of concerns under "setup flexibility."

4 Core Beliefs

Our proposal is based on two core beliefs that we share below.

Belief I: We can optimize for at most 3 of the 4 design goals of PPSCs (given the limitations of current cryptography).

We believe that security and privacy are non-negotiable. We want to optimize our system for flexibility so that we can support arbitrary PPSCs. That leaves us with compromising for efficiency to some degree.

Before we state Belief II, we provide some motivation to the reader.

When we look at previous attempts to construct confidential transactions, a clear pattern emerges. We often see what we call "additively homomorphic" encryption or commitment schemes in which:

$$\mathsf{Enc}(a) + \mathsf{Enc}(b) = \mathsf{Enc}(a + b) \tag{1}$$

Assume everything is well-defined and a, b are valid inputs to the $\mathsf{Encryption}$ function. These additively homomorphic encryption/commitment schemes are generally used to perform confidential transactions in the following manner:

$$\mathsf{Enc}(\mathsf{balance}) + \mathsf{Enc}(\mathsf{transfer}) = \mathsf{Enc}(\text{balance after transfer})$$

Thus, neither the user's balance nor the transfer amount sent is ever revealed in the clear. While homomorphic addition is satisfactory for performing confidential transactions, it limits the potential use cases when we look towards representing arbitrary computations. We say an encryption scheme is "multiplicatively homomorphic" if the following holds:

$$\mathsf{Enc}(a) \cdot \mathsf{Enc}(b) = \mathsf{Enc}(a \cdot b) \tag{2}$$

The reader may ask "but isn't multiplication just repeated addition?" Consider the following example where we attempt to add the encryption of a (i.e. $\mathsf{Enc}(a)$) to itself c times:

$$\underbrace{\mathsf{Enc}(a) + \mathsf{Enc}(a) + ... + \mathsf{Enc}(a)}_{c \text{ times}} = c \cdot \mathsf{Enc}(a)$$

Thus, c behaves as a constant and is a "public" value that others can see in the clear. Depending on the specific use case, this may or may not be an issue. However, by only allowing for *additively homomorphic* encryption schemes, we are limiting the range of use cases we can represent from the outset.

Suppose we want to multiply two users' balance amounts together—call them a and b (for user A and user B respectively). With homomorphic addition, we would have to do one of the following:

$$\underbrace{\mathsf{Enc}(a) + ... + \mathsf{Enc}(a)}_{b \text{ times}} = b \cdot \mathsf{Enc}(a)$$

in which we add $\mathsf{Enc}(a)$ to itself b times or we could do the reverse (add $\mathsf{Enc}(b)$ to itself a times):

$$\underbrace{\mathsf{Enc}(b) + ... + \mathsf{Enc}(b)}_{a \text{ times}} = a \cdot \mathsf{Enc}(b)$$

In either case, one of the values (either balance a or balance b) is revealed in the clear to the public!

Instead, with a multiplicatively homomorphic encryption scheme, we could perform the following:

$$\mathsf{Enc}(a) \cdot \mathsf{Enc}(b) = \mathsf{Enc}(a \cdot b)$$

in which neither user's balance (a nor b) is ever revealed in the clear. If we want an encryption scheme that is both additively homomorphic and multiplicatively homomorphic, we want an encryption scheme that is *fully homomorphic.* We are now ready to state our final belief.

Belief II: FHE is always naively introduced in the context of outsourcing computation to an "all-powerful cloud." However, FHE's

true strength lies in viewing it as an extension of public key cryptography (particularly in the distributed computing setting).

5 Our Design Goals for PPSCs

We will outline our design goals in terms of the 4 properties we described above—optimizing for security, privacy, and flexibility (at the cost of losing some efficiency).

Security Assumptions

We want the security assumptions of our system to be based solely on cryptography. We do not want to rely on hardware or game-theoretic assumptions for security since we want provable security. We prefer to use well-understood cryptographic hardness assumptions wherever possible. Additionally, we avoid using trusted setups both for security and flexibility reasons.

Privacy

Our scheme will provide confidentiality. We will not focus on anonymity at this time; however, we will leave open the door for possible exploration of it in the future. The confidentiality of our scheme will be based solely on cryptography and not game-theoretic assumptions.

Flexibility

Our scheme will provide flexibility by allowing users to model arbitrary computations in a privacy-preserving manner through the use of fully homomorphic encryption schemes. We will use universal NIZK with no trusted setup. Additionally, our system should use an account-based model to support stateful computations.

Semi-Efficiency

We will avoid using trusted setups (even though they generally allow for the fastest verification times and smallest proof sizes). SNARKs (non-universal) would require us to go through the costly trusted setup process each time we wanted to prove a new statement. While SNARKs make sense when we're attempting to prove the same statement many times (as Zcash does), it does not make sense when we're trying to provide flexibility to users to prove arbitrary statements [33]. Trusted setups can also be viewed as being in conflict with providing security based purely on cryptography.

We want our scheme to be accessible to as many individuals as possible; to accomplish this, we must eliminate the need for expensive hardware for simple use cases. Thus, STARKs are not a viable option (in addition to having large proof sizes) [6]. This leaves us with Bulletproofs which provide moderate proof sizes, and moderate verification times (particularly when batched together for better efficiency) [10].

6 Importance of Flexibility

As no private computation scheme has yet optimized for flexibility, security, and privacy, the reader may ask why now do so? After all, there may be good reason to prioritize efficiency over flexibility. When asked about the biggest problem facing blockchain today, almost everyone immediately thinks of scalability (which goes in hand with efficiency).

We are not looking to just build a widely-traded private coin (such as Zcash or Monero) that acts in place of fiat currency [33, 2]. We are looking to build a platform which supports the development of arbitrary private applications (via smart contracts). Our mission is closest to that of Ethereum's with the caveat that we believe privacy is of crucial importance.

As we've seen, privacy is difficult to achieve on Ethereum in its present state. Confidential transactions are currently possible on Ethereum (for example, through Zether or AZTEC) [9, 32]. However, more complex confidential PPSCs are currently impossible to construct there.

By optimizing for flexibility at layer 1, we leave open the possibility for layer 2 applications that can optimize for efficiency for specific use cases. Our platform will optimize for security (like Bitcoin), privacy (like Zcash), and flexibility (like Ethereum)—which together is a novel combination.

7 Cryptographic Primitives

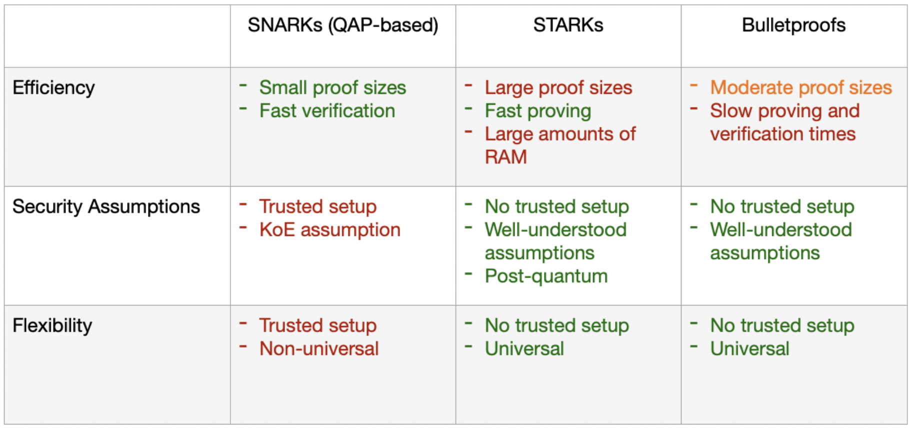
The two main cryptographic primitives we'll rely on for constructing our PPSCs are Bulletproofs and FHE [10, 17]. Below, we'll provide a quick review of the two (mainly a summary from the papers) and their relevance to PPSCs.

7.1 Bulletproofs

We choose Bulletproofs as our non-interactive ZKP system. Although Bulletproofs don't offer the fastest verification times, they can be "batched together" for better efficiency [10, 9]. Additionally, they offer moderate proof sizes—approximately 1-3kB/proof which is 1 order of magnitude larger than SNARKs and 1-2 orders of magnitude smaller than STARKs [3].

Bulletproofs have no trusted setup process (unlike SNARKs), do not require costly hardware (unlike STARKs), and are based on well-understood hardness assumptions (discrete logs) [33, 6, 10].

Below, we provide a chart comparing the 3 major ZKP systems in terms of the 4 design goals we previously defined. We omit any comparison of "privacy" as all ZKP systems by definition provide confidentiality. None of them provide any level of anonymity out of the box (unless combined with other tools such as ring signatures or tumblers).



	SNARKs (QAP-based)	STARKs	Bulletproofs
Efficiency	- Small proof sizes - Fast verification	- Large proof sizes - Fast proving - Large amounts of RAM	- Moderate proof sizes - Slow proving and verification times
Security Assumptions	- Trusted setup - KoE assumption	- No trusted setup - Well-understood assumptions - Post-quantum	- No trusted setup - Well-understood assumptions
Flexibility	- Trusted setup - Non-universal	- No trusted setup - Universal	- No trusted setup - Universal

Figure 1: Comparison of ZKP systems.

7.2 Fully Homomorphic Encryption: BGV Scheme

FHE allows for arbitrary computations on ciphertexts in a structure-preserving way without the need for a private key [17]. Recall that any computation can be represented via addition and multiplication—both of which FHE supports.

All current constructions of FHE rely on a type of post-quantum cryptography called lattice-based cryptography. However, different schemes use spaces with varying amounts of structure—ranging from lattices in $\mathbb{R}^n$ to more structured lattices using polynomial rings.

Additionally, FHE can be grouped into 3 different categories based on how it models computation—boolean circuits vs. modular arithmetic vs. approximate number arithmetic (see [12] for details). We favor the use of a scheme that models computation as modular arithmetic for our use case. The main schemes that fall under this category are the BV scheme and its variants (BGV, FV) [8, 15]. These schemes also support an optimization called "ciphertext packing" or "batching" which allows for "SIMD-style" computations [31].

Note: Post-quantum cryptography is currently undergoing a standardization process by NIST. Of the 17 PKE/KEM candidates under consideration, 9 of them are lattice-based. Of the 9 candidate signature schemes under consideration, 3 of them are lattice-based. NIST expects to have a draft of the standard published by the end of 2022. [25, 29]

A Review of the BGV Scheme

The BGV (Brakerski-Gentry-Vaikuntanathan) scheme is a FHE scheme that improves upon the BV scheme [8]. It comes in both a ring and non-ring version but we will focus our attention on the more efficient ring version.

The BGV scheme is a leveled FHE scheme, meaning that we can only perform a certain number of multiplications sequentially before we reach a point at which we can no longer decrypt the resulting ciphertext [8]. We can use "bootstrapping" as an optimization to avoid having to specify the number of "levels" (of arithmetic circuits) in advance. Each level also has its own set of (public and secret) keys [8]. Recall that bootstrapping involves applying the decryption function to a ciphertext homomorphically to reduce the amount of noise - thus, "refreshing" the ciphertext [8]. Bootstrapping, however, is an expensive operation (even more so than homomorphic multiplication).

We can successfully decrypt a ciphertext obtained from this scheme if its noise is "small enough." However, each time we perform homomorphic operations (especially multiplication), the ciphertext's noise grows! To prevent the noise from growing so large such that decryption fails, a technique called "modulus switching" (which can be performed by anyone) is used to keep the noise level roughly constant.

When we multiply two ciphertexts c (decryptable under secret key s) and c' (decryptable under secret key s') together, we get a long resulting ciphertext that is decryptable under a long secret key. Having to work with these very long keys and ciphertexts impacts the efficiency of our scheme so we utilize an additional technique called "key switching" that instead allows us to work with a smaller ciphertext and smaller secret key in place of the originals [8].

Note: The authors of the paper avoid assuming circular security (i.e. what happens when you ask a system to encrypt its own key) and therefore specify a different key for each level of the scheme. If we're willing to assume circular security, the same key may be used for each level [8]. For more information on circular security, see Matt Green's post ("Wonk post: Circular Security") on the topic [18].

Sitting inside the BGV scheme is the standard Ring-LWE encryption scheme which naturally supports homomorphic addition [24]. Below, we cover the main functionalities of the BGV scheme as in the original paper [8].

1. $\mathsf{Setup}(1^\lambda, 1^\mathrm{L}) \rightarrow \{params_j\}$: Setup takes as inputs the security parameter and the number of levels L. It then outputs the parameters $params_j$ for each level j - which includes a modulus, noise distribution, and an integer. We also obtain a ladder of decreasing moduli that will be used in the modulus switching procedure in the algorithm called "$\mathsf{Refresh}$" in 7.

2. $\mathsf{KeyGen}(\{params_j\}) \rightarrow (pk, sk)$: KeyGen takes as inputs the parameters $\{params_j\}$ obtained from Setup. It outputs the secret key sk which consists of the secret keys s_j for each level j, the public key pk which consists of the public keys pk_j for each level j, and the auxiliary information needed to facilitate the key switching procedure in 7. Recall that there is a set of keys for each level.

3. $\mathsf{Encrypt}(params, pk, m) \rightarrow c$: $\mathsf{Encrypt}$ takes as inputs the scheme's param-

eters *params*, a message m in the plaintext space and the public key pk for the appropriate level. It outputs a ciphertext c.

4. $\mathsf{Decrypt}(params, sk, c) \rightarrow m$: Decrypt takes as inputs the scheme's parameters *params*, the appropriate secret key sk (dependent on the level), and a ciphertext c. It outputs the corresponding plaintext m.

5. $\mathsf{HomAdd}(pk, c_1, c_2) \rightarrow c''$: HomAdd takes as inputs two ciphertexts c_1, c_2 from the same level and the public key pk. It runs the Refresh procedure described in 7 and finally outputs the resulting (summed) ciphertext c''.

6. $\mathsf{HomMult}(pk, c_1, c_2) \rightarrow c''$: HomMult takes as input two ciphertexts c_1, c_2 from the same level and the public key pk. It uses the Refresh procedure of 7 and finally outputs the resulting (product) ciphertext c''.

7. $\mathsf{Refresh}(c, \tau, q_j, q_{j-1}) \rightarrow c'$: Refresh takes as input a ciphertext c, the auxiliary information τ to facilitate key switching (from the KeyGen algorithm), the current modulus q_j and the next modulus q_{j-1}. It then does the following:

   (a) "Expands": Expand the ciphertext into a powers-of-2 representation.
   (b) "Switch Moduli": Scales the ciphertext according to the new modulus.
   (c) "Switch Keys": Performs the key switching procedure resulting in a new ciphertext decryptable under a different key and modulus.

   Refresh finally outputs a ciphertext c'.

A potential optimization for the BGV scheme is "batching" (based on the Chinese Remainder Theorem) which allows us to evaluate the function f on ℓ blocks of encrypted data more efficiently than is possible when evaluating the function f on a single block of data ℓ times [8, 31].

8 Challenges and Recent Advances

In building a PPSC system that combines ZKPs with FHE, we run into some challenges. We outline the two major challenges that arise and our solutions to them.

Challenge I: How do we combine FHE (which uses lattice-based cryptography) with Bulletproofs (which use discrete logarithms and elliptic curves)?

Thanks to recent work from Del Pino et al. in "Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts," we know how to efficiently prove properties about ciphertexts (from LBC) using ZKPs based on discrete logarithms [13]. Specifically, the authors show to combine Bulletproofs with Ring-LWE based encryption schemes. We will also use ideas from “Zether: Towards Privacy in a Smart Contract World” to further aid us [9].

Challenge II: FHE isn’t very efficient and can sometimes require expensive hardware to achieve acceptable performance. Can we get around this issue?

It is necessary for us to distinguish between performing addition vs. multiplication in our FHE scheme. Addition is a relatively cheap operation and causes little issues with noise management whereas multiplication is an expensive operation that causes the noise to grow a lot.

Recall that confidential transactions can be modelled solely in terms of homomorphic addition. Other common smart contracts can also be modelled solely in terms of homomorphic addition—sealed bid auctions, voting [9].

As stated earlier, we want our scheme to be accessible to as many individuals as possible; to accomplish this, we must eliminate the need for expensive hardware for acceleration. Thus, we will avoid using (homomorphic) multiplication unless absolutely necessary. The individual will be able to perform confidential transactions and participate in simple smart contracts without needing access to any specialized hardware. However, some may need access to some specialized hardware to perform homomorphic multiplications efficiently.

The hope is to make our scheme and its main functionalities accessible to all, although we recognize that this may not be the perfect solution.

9 Possible Lines of Work

In this section, we explore two possible lines of PPSC research using Bulletproofs. Ultimately, we choose to pursue the 2nd proposal inspired by “Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts” since we hope to successfully incorporate FHE into a PPSC scheme of our own. We have included the 1st proposal since our scheme will draw some inspiration from Zether’s construction.

Note: The edited contents of this section originally belonged to a longer document written in July 2019 exploring two possible lines of research. Some of the information here is no longer up to date with our current work.

9.1 Proposal 1: Using “Zether”

We first introduced Zether in the document “ZKPs for Privacy-Preserving Smart Contracts and Transactions.” Keep in mind that the authors implement Zether using the curve bn-128 as it provides the lowest gas cost. JP Morgan has recently introduced their own reference code for the fully anonymous version of Zether [26]. Their technical report provides helpful details in terms of understanding the proof protocol for anonymous Zether (recall that the original paper only provided a brief outline of anonymous Zether), asymptotic runtimes, proof sizes, and gas costs [14].

Ethereum Compatibility

Zether was designed to be compatible with Ethereum though it presents some implementation issues. As is, we can implement Zether in Ethereum and conduct confidential transactions (which cost $\sim$7.2 million gas using curve bn-128) [9]. However, we cannot implement the anonymous version of Zether as is since the cost would greatly exceed the $\sim$8 million gas limit [9]. Furthermore, due to Ethereum limitations to curve selection, we cannot use a more secure curve [9].

Improvements and Optimizations

As seen previously, Zether is incredibly costly in terms of gas/verification time. However, we can look into batching transactions together to reduce the overall cost per transaction [10, 9]. We can also consider bridging between Ethereum's blockchain and the Zether one using some intermediary.

Roughly speaking, to solve for the Ethereum compatibility issue, we *could* use an intermediary blockchain (such as Cosmos) to move between a newly designed Zether blockchain and the Ethereum network so that we're no longer bound by the EVM. Cosmos refers to this as "bridging" [28]. However, it's not clear how the Cosmos hub would handle the more complex logic involved in private smart contracts [23]. By going through the Cosmos hub and pooling ETH/ZETH belonging to many different users together, we may be able to achieve better privacy guarantees while still achieving some form of compatibility with Ethereum.

Challenges

There are a number of challenges in developing and implementing Zether.

Anonymous Zether. In using the anonymous version of Zether, the user involved in the transaction must choose a set of users in which to hide himself in a "one-out-of-many proof" (which can also be used to instantiate a ring signature scheme; see [19] for the construction) [9]. However, how to pick this set and the optimal set size are contentious issues. Also, how would we maintain anonymity if we need to pay miners' fees in ETH (if interested in implementing this on Ethereum)?

Let the anonymity set be denoted by A_s. Unfortunately, anonymous Zether has incredibly high gas costs (recent estimates of $\sim$31.5 million gas for an $|A_s| = 8$ and an astounding 109 million for an $|A_s| = 32$) [14, 26]. This behavior is worse than that of state-of-the-art ring signatures (see [5]) which have proof sizes that grow logarithmically with the ring size.

Behavior. Even if the anonymous version of Zether had a low enough gas cost to deploy on Ethereum, Zether would likely suffer from some of the same privacy issues as Zcash does in sending coins between “shielded” and “transparent” addresses [21]. We emphasize that this is not a “Zcash” specific issue; rather, allowing for interaction between public and private accounts/addresses introduces some leakage of information.

Additionally, If we were to implement Zether on a new blockchain, we would run into the same anonymity issues as Monero when picking a “ring” of users [27]. Ring signatures are one of the main cryptographic techniques for achieving anonymity but the cost and privacy level achieved in a real-world setting can be concerning.

Some Thoughts

Arguably, the greatest strength of Zether is its potential for creating privacy-preserving smart contracts that are compatible with Ethereum. Recall that the primary issue with implementing Zether on Ethereum is just how expensive the elliptic curve operations are (taking up to 90 percent of the total gas cost) [9]. Our Zether blockchain could be based off Ethereum’s but with further optimizations made for elliptic curve operations and support of “better” curves. Even then, how would we account for the astronomical cost of anonymous proofs?

As briefly alluded to above, it may be possible to find some way to bridge between two blockchains (Ethereum and a Zether specific one we’d have to create). This might allow us to implement the fully anonymous version of Zether while still achieving some weak form of compatibility with Ethereum. However, this will be technically challenging to implement securely and provides questionable value for the end user. To shorten the development timeframe, we’d have to rely on another network (such as Cosmos) to serve as the intermediary between the two blockchains. This naturally means tailoring our Zether blockchain to work specifically with the intermediary’s IBC protocol [28].

9.2 Proposal 2: Using “Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts”

We first encountered this paper in the“ZKPs for Privacy-Preserving Smart Contracts and Transactions” document under the section “On Integrating FHE.” After further consideration, we believe the techniques and ideas presented in this paper could allow us to perform confidential transactions and potentially even build privacy-preserving smart contracts.

Recall that in the paper “Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts,” the authors propose using a new variant of Bulletproofs to prove that FHE ciphertexts are “well-formed” [13]. This is potentially useful in the distributed public ledger setting where we have many different users encrypting their plaintext under the same encryption key (and whoever is performing the computation wants to ensure that a misbehaving user isn’t sending him some malformed data that will corrupt the entire computation). However, this may also be important when we have a single user and the computing party wants to be confident that the user has provided well-formed data before performing a costly computation. We refer to these problems as the "misbehaving user FHE problem."

We see this line of research as more promising than that of Zether. It allows us to further expand our FHE product range. Even more, it provides value to the company before we build a system that allows for confidential transactions or PPSCs.

Overview of Usage

We'll first provide a brief technical overview of the ideas presented in the paper. A Ring-LWE ciphertext can be expressed as the matrix-vector equation (we italicize vectors and bold matrices):

$$\mathbf{A}\vec{s} = \vec{t} \tag{3}$$

where $\mathbf{A}$ is the public key, $\vec{s}$ is the message and randomness, and $\vec{t}$ is the ciphertext. We assume all operations are performed over the polynomial ring $\mathbb{R}_q = \mathbb{Z}_q[X]/ < f >$ where q is an integer and f is a monic irreducible polynomial of degree d in $\mathbb{Z}[X]$. The Prover can commit to the coefficients in $\vec{s}$ using a Pedersen commitment $c = \mathsf{COM}(\vec{s})$ and can then prove in zero-knowledge that the coefficients (interpreted as the polynomial vector $\vec{s}$) satisfy the above matrix-vector equation. The authors note that since we have a Pedersen commitment c to the coefficients of $\vec{s}$, we can use (standard) Bulletproofs to prove additional properties about the plaintext if so inclined [13].

The hope is that by using a scheme supporting homomorphic computation, we can represent a greater variety of privacy-preserving applications (and thus support arbitrary PPSCs) than by just using a scheme that is "weakly homomorphic" (such as Zether which uses ElGamal encryption which supports only additive homomorphic operations). We'd almost certainly have to develop our own blockchain.

Proposal:

- Each user will generate his own set of keys (with a Ring-LWE based scheme). He'll use his key to encrypt his balance and any other information he wants to remain private. As above, we can hopefully make this work with a (Ring-LWE based) FHE scheme.

- To perform a confidential transaction, user A might encrypt the amount being sent under his own key and that of the receiver (WLOG say user B). He'll prove in ZK well-formedness of the ciphertext, conditions ensuring that his remaining balance is non-negative, that the amount being sent is positive, etc.

Challenges

The paper ("Short Discrete Log Proofs for FHE and Ring-LWE Ciphertexts") was not written with PPSCs in mind and thus we would have to do significantly more research than Proposal 1 (based on Zether). Some of the major obstacles we face for Proposal 2 is lack of Ethereum compatibility implying the development of our own blockchain, significant amount of research needed for the architecture of such a system, and concerns over possible key reuse attacks.

Beyond Ethereum. It's unlikely that a confidential payment mechanism based on Ring-LWE will be feasible with Ethereum. However, if it is possible to build arbitrary PPSCs via some of the ideas here, that may present enough value to justify the creation of a new blockchain.

Required Research. Significantly more research needs to be done for Proposal 2—what would the architecture of such a system look like, what's the efficiency like, can we successfully integrate FHE into the system and how will that affect the proofs and ciphertext sizes, what would our blockchain look like if we need to develop our own?

Integration of Different Bulletproofs. To create standard Bulletproofs (which allow a Prover to prove that his secret value lies in some range), the Prover needs to use an inner-product argument that shows he knows two vectors that satisfy a given inner-product relation. The original Bulletproofs paper does *not* use an inner-product argument that's ZK [10]. This leads to additional complexity in the range proof since the Prover will need to blind some of the vectors [13]. Refer to Sections 3 and 4 of [10] for further details. This paper, on the other hand, introduces a new variant of Bulletproofs that uses an inner-product proof that *is* ZK [13]. The authors accomplish this by using a Schnorr-type proof. By using a inner-product proof that's ZK, the ZK range proof is shortened to 3 rounds instead of the usual 5 since we no longer have to blind the vectors. For further technical details on their Bulletproofs variant, see Section 4 of [13].

Key Reuse Attacks. Ring variants of lattice-based encryption schemes are more efficient than the corresponding non-ring versions. The intuitive explanation for this difference is that the additional structure of the ring allows for compression and thus efficiency. However, the additional structure provided by the ring can also be exploited to attack the scheme. A number of attacks have been discovered for various Ring-LWE based schemes. What does that mean for us if we want to use a Ring-LWE based encryption scheme? Is there something fundamentally insecure about using Ring-LWE?

A particularly illuminating paper covering attacks on Ring-LWE based schemes is "How (Not) to Instantiate Ring-LWE" from Chris Peikert [30]. Peikert argues that every insecure instantiation of a Ring-LWE based scheme suffers from the same issue—namely that the error distribution was chosen incorrectly and is of a different shape than the one supported by the actual hardness proof.

More specifically, Peikert reviews all known attacks on Ring-LWE and finds that these insecure instantiations use a non-dual form of Ring-LWE along with spherical error distributions which are not "well spread" enough relative to the ring [30]. Even the insecure schemes allow for a provably hard instantiation with a different (correctly chosen) error distribution.

The paper also goes on to give sufficient conditions to guarantee that a Ring-LWE instantiation is provably immune to generalizations of the attacks (reduction to errorless LWE so that the search version is trivially solvable or reduction modulo an ideal divisor q of the modulus qR so that the decision version can be solved if certain conditions on q and the error distribution are also satisfied) used on the schemes shown to be insecure (see Section 3 "Attack Framework") [30] .

In light of these findings, we do not believe that we should be overly concerned about the security of Ring-LWE.

Potential Upsides

This line of research offers two upsides that we cannot achieve from any Zether-based proposal—namely we have (limited) resistance to quantum attacks and we can possibly integrate FHE into our PPSCs.

Limited Quantum Resistance. By using an encryption scheme based on Ring-LWE (or lattices more generally speaking), we obtain some amount of resistance against quantum attacks. Specifically, we get that the secrecy of our scheme is based on the hardness of Ring-LWE; however, the soundness of the ZKPs is based on the discrete log problem [13].

Arguably, the biggest fears around quantum computers are "store-now-decrypt-later" attacks in which an attacker stores encryptions now with the goal of decrypting them later in the future when he has access to a powerful enough quantum computer. We, however, would not be vulnerable to store-now-decrypt-later attacks with our setup.

FHE Integration. As this proposal is based on using Ring-LWE, we naturally leave open the possibility of integrating FHE or even multi-key FHE with our product (when such a scheme is somewhat efficient). Current PPSCs and confidential payment schemes rely on additively homomorphic commitment schemes or additively homomorphic encryption schemes. By using FHE, we're not restricted to addition in the types of computations we support.

Some Thoughts

Other private transaction schemes/PPSCs attempt to exploit additively homomorphic encryption schemes or additively homomorphic commitment schemes to achieve their goal. However, this design choice inherently limits the types of computations that can be represented. By using a FHE scheme as our foundation, we aren't inherently limited in this way.

As we'd need to use a Ring-LWE based scheme, this proposal is not immediately compatible with nuFHE so we'd have to use/fork a different library. The timeline here is much longer than Proposal 1. We'd have to figure out the architecture of the scheme and understand how to securely and efficiently combine blockchain with FHE and Bulletproofs.

We can mitigate some of the risk involved in this proposal by offering a standalone product in the FHE space. However, we are positioning ourselves as believing in the increased usage/growth of FHE.

9.3 Bulletproofs' Considerations

Regardless of the specific scheme/ideas we choose to build upon (Zether or Short Discrete Log Proofs for FHE/Ring-LWE Ciphertexts), we face a few challenges in implementing Bulletproofs [9, 13]. Currently, there are only two major libraries to choose from—both of which are tailored to work with specific elliptic curves and are not production ready [20, 1].

Recall also that the biggest bottleneck of Bulletproofs is their slow verification time (at least compared to that of zk-SNARKs and zk-STARKs). Thus, batch verification is arguably an important optimization to consider when implementing Bulletproofs. Concretely, Bulletproofs are about two orders of magnitude slower than zk-SNARKs when it comes to verification (SNARKs take ∼10ms to verify whereas Bulletproofs take ∼1100ms) [3].

Dalek

A well-known implementation of Bulletproofs is dalek (see [20]). It comes from Henry de Valence, Cathie Yun, and Oleg Andreev at Interstellar and is written entirely in Rust. It implements Bulletproofs using Ristretto with Curve25519 which is not currently supported by Ethereum.

Secp256k1

Another popular Bulletproofs library is secp256k1 (see [1]) from Andrew Poelstra at Blockstream. The library is written entirely in C and implements Bulletproofs using curve secp256k1 (as could be guessed from the name). There is minimal documentation available for this library.

10 Overview of Architecture

Our scheme will take inspiration from Zether as we believe it comes closest to achieving our goals (bar total flexibility). Roughly speaking, we wish to replace the additively homomorphic encryption scheme in Zether (they use the ElGamal encryption scheme) with that of a Ring-LWE public key encryption scheme [9, 24, 13]. This Ring-LWE encryption scheme will then be extended to a FHE scheme to provide total flexibility [24, 8, 13]. We also wish to perform confidential transactions directly without the use of a smart contract. This gives rise to a larger class of transactions (similar in flavor to Zcash's 4 types) [33].

A Brief Review of Confidential Transactions and PPSCs

While there are many schemes—both applied and theoretical—that offer the power to to perform confidential transactions (Zcash, Monero, Aztec, Quisquis, Zether, Zexe, etc.), there are very few that offer the ability to create smart contracts in a privacy-preserving manner [32, 2, 9, 7, 22, 33].

Arguably, the first requirement we impose on ourselves is that the scheme uses an account-based model that is able to maintain some notion of "state." Thus, we eliminate all UTXO-based models. We could consider combining the account model with that of UTXOs, but this introduces unnecessary complexity [16].

One of the few proposals in the space that uses an account-based model is Zether [9]. Zether combines an additively homomorphic encryption scheme (ElGamal) with Bulletproofs to allow for confidential transactions. While Zether does not support arbitrary computations, their scheme was designed as a solution for Ethereum [9]. Their work serves as one of the main sources of inspiration for our proposal.

Flexibility

We differentiate between users based on their use of homomorphic multiplication.

Some users may use *only* homomorphic addition. Suh a user is able to perform confidential transactions and participate in simple private smart contracts like voting and sealed-bid auctions (i.e. any private smart contract that does not require him to perform homomorphic multiplications) [9].

Other users may use both homomorphic addition and homomorphic multiplication (i.e. the full range of homomorphic computations). Such a user is able to perform confidential transactions and participate in all private smart contracts. The main caveat is he needs additional physical resources to do so for efficiency reasons.

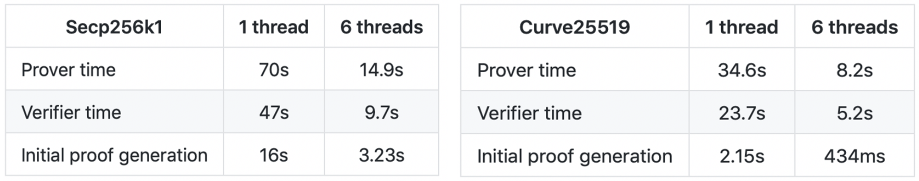
Implementing FHE (securely and correctly) requires some degree of mathematical competency.

We might consider restricting all users to only making use of homomorphic addition initially.

11 Implementation

Bogdan built a proof-of-concept from the "Short Discrete Log Proofs for FHE and Ring-LWE Ciphertext" paper covering the verifiable encryption scheme described in section 1.5 [13]. The authors had not provided any code so we were interested in seeing how the scheme performed in practice with popular curves.

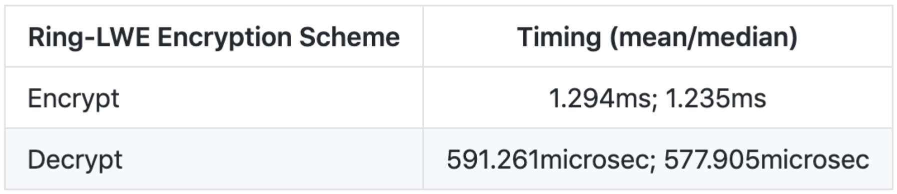
The preliminary results are based off an implementation in a JIT interpreted language for iteration purposes. This prototype is not secure and should not be used in production.



Secp256k1	1 thread	6 threads
Prover time	70s	14.9s
Verifier time	47s	9.7s
Initial proof generation	16s	3.23s

Curve25519	1 thread	6 threads
Prover time	34.6s	8.2s
Verifier time	23.7s	5.2s
Initial proof generation	2.15s	434ms

Figure 2: PoC of [13] using secp256k1 and curve25519.



Ring-LWE Encryption Scheme	Timing (mean/median)
Encrypt	1.294ms; 1.235ms
Decrypt	591.261microsec; 577.905microsec

Figure 3: Ring-LWE encryption.

Our estimates (seen in Figures 2, 3) come from an interactive version of the protocol. The paper estimates that the size of the proof would be around 1.3kB so we do not include this information.

The measurements were taken on an Intel i7 at 2.6 GHz without any specialized hardware. We plan on using Curve25519 but have included secp256k1 as it's a popular curve in the cryptocurrency space. For more details, please see our Github repo (LogProof).

12 References

[1] libsecp256k1. `https://github.com/apoelstra/secp256k1-mw/tree/bulletproofs`.

[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`, dec 2019.

[5] Michael Backes, Nico Döttling, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Ring signatures: Logarithmic-size, no setup—from standard assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–311. Springer, 2019.

[6] 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.

[7] 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.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277, 2011. `https://eprint.iacr.org/2011/277`.

[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] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 185–200. IEEE, 2019.

[12] Homomorphic Encryption Standardization Consortium. Building applications with homomorphic encryption. Available at `http://homomorphicencryption.org/wp-content/uploads/2018/10/CCS-HE-Tutorial-Slides.pdf`, 2018.

[13] 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.

[14] Benjamin Diamond. Anonymous zether. Available at `https://github.com/jpmorganchase/anonymous-zether`, 2019.

[15] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[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] Craig Gentry. *A fully homomorphic encryption scheme*, volume 20. Stanford University, 2009.

[18] Matthew Green. Wonk post: Circular security. A Few Thoughts on Cryptographic Engineering, 2012. `https://blog.cryptographyengineering.com/2012/04/27/wonk-post-circular-security/`.

[19] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 253–280. Springer, 2015.

[20] Cathie Yun Henry de Valence and Oleg Andreev. Bulletproofs. Github, 2018. `https://github.com/dalek-cryptography/bulletproofs`.

[21] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in zcash. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 463–477, 2018.

[22] 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.

[23] Jae Kwon and Ethan Buchman. Cosmos: A network of distributed ledgers. Available at `https://cosmos.network/cosmos-whitepaper.pdf`.

[24] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013.

[25] Dustin Moody. The 2nd round of nist pqc standardization. NIST, 2019. `https://csrc.nist.gov/CSRC/media/Presentations/the-2nd-round-of-the-nist-pqc-standardization-proc/images-media/moody-opening-remarks.pdf`.

[26] JP Morgan. Anonymous zether. Available at `https://github.com/jpmorganchase/anonymous-zether`, 2019.

[27] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, et al. An empirical analysis of traceability in the monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2018(3):143–163, 2018.

[28] Cosmos Network. Inter-blockchain communication protocol. Available at `https://github.com/cosmos/ics/tree/master/ibc`, 2019.

[29] NIST. Nst pqc standards next steps. NIST, 2019. `https://csrc.nist.gov/CSRC/media/Presentations/nist-pqc-standards-next-steps/images-media/liu-nist-pqc-next-steps-2019.pdf`.

[30] Chris Peikert. How (not) to instantiate ring-lwe. In *International Conference on Security and Cryptography for Networks*, pages 411–430. Springer, 2016.

[31] N.P. Smart and F. Vercauteren. Fully homomorphic simd operations. Cryptology ePrint Archive, Report 2011/133, 2011. `https://eprint.iacr.org/2011/133`.

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

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

[34] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guranteed privacy. Enigma, 2015. `https://enigma.co/enigma_full.pdf`.