Snakepath: A Lightweight Protocol for Encrypted Crosschain Communication

Benjamin Simon and Leor Fishman

September 2022

1 Introduction

When the internet was invented in the 1980s, the Hypertext Transfer Protocol (HTTP) became the first widely-adopted protocol for web data communication. HTTP facilitated all kinds of online activity, but it did so in a way that was public and un-encrypted. As a result, individuals that were brave (or foolish) enough to send private data over the web often had their data compromised. Credit card data, health care records, social security numbers—none of these were safe on the early internet.

In 1994, Netscape created a new variant of the data transfer protocol: HTTPS, the "S" standing for secure. HTTPS uses an encryption method—first SSL, now TLS—to secure data.[1] The advent of HTTPS was a watershed moment for the internet, unlocking previously unserviceable use cases that required sensitive data. Today, HTTPS remains a bedrock protocol for web 2.0, used by the vast majority of all websites.

There are strong parallels between the web before HTTPS and today's smart contract blockchains. These blockchains, most notably Ethereum, have proven to be viable base layers for distributed financial activity. But the activity that blockchains facilitate is still limited—and not just by scalability constraints or user experience friction.

Smart contract blockchains like Ethereum are fully transparent. All current and past historical data are publicly viewable and auditable. This transparency is a core feature of blockchains. It enables them to function trustlessly, and it also allows onchain applications to compose with each other seamlessly.

The blockchain's transparency is part of what makes it such a promising alternative to traditionally-closed digital systems. For example, in Decentralized Finance (DeFi)—the suite of permissionless financial applications built on smart contract blockchains—leverage is trackable onchain, liquidity is universally accessible, interest rates are programmatic, and there is no such thing as a "hidden fee."

[1] Transport Layer Security, or TLS, is an improved version of SSL (Secure Sockets Layer).

But public data is a double-edged sword. While it provides tangible benefits for some applications, it also severely curtails the kinds of applications that can be built onchain, as these chains cannot privately process sensitive data. Much like the early web was hamstrung before HTTPS, smart contract blockchains today support only those applications that do not require Personal Identifiable Information (PII) or other sensitive data.

The world of useful smart contract applications would expand dramatically if it were possible for these applications to securely process and compute over sensitive data. (See Section 4 for our discussion of potential applications that leverage privacy-preserving computation that can be built using Snakepath.)

In this paper, we outline the Snakepath protocol, a critical building block for bringing private data onchain in a useful yet privacy-preserving manner.

Snakepath is a protocol for encrypted inter-blockchain communication. More specifically, Snakepath enables public chains to call arbitrary functions on private compute chains while preserving the privacy of the inputs and validity of the outputs. Snakepath is built using a primitive that we call TNLS ("Transport *Network* Layer Security") which is effectively a blockchain derivative of the TLS protocol.

Snakepath itself does not store or compute over data. Rather, it connects public blockchains and their applications to privacy-preserving computation networks. This design allows public blockchain applications to build and operate private computation contracts on privacy-preserving chains while keeping their primary smart contract logic and liquidity on public blockchains.

Ultimately, Snakepath enables the building of new applications that combine the transparency, UX, and latency benefits of public blockchains with the trust-minimized and private computation features of privacy-preserving blockchains.

2 Computational Privacy

Before delving into Snakepath in greater detail, we will first examine the existing infrastructure landscape for computational privacy. This review is important because the entire objective of Snakepath is to enable public blockchain applications to make use of trust-minimized privacy-preserving computation infrastructure to securely and privately process offchain sensitive data.

With respect to computational privacy, we will first review the primary types of infrastructure and assess briefly their privacy, security, generality, and performance tradeoffs (section 2.1). We then explore how these privacy-preserving computation technologies can be, and have been, applied to enable privacy-preserving computation for blockchain applications (section 2.2).

We find that, while there are a variety of protocols that offer various types of computational privacy, there is no existing solution for generic and trust-minimized computational privacy for public blockchain applications. Existing privacy-preserving computation protocols that fit the bill are not currently interoperable with public smart contract blockchain applications.

2.1 Types of Computational Privacy Infrastructure

For the purposes of this paper, we define computational privacy as "computations that do not leak information about a system's data to anyone."

There are a variety of computational privacy solutions. As we will see, there are different levels of privacy and different trust assumptions depending on the computational privacy technique employed. In this subsection, we review four common types of infrastructure related to computational privacy: Zero Knowledge Proofs, Trusted Execution Environements, Homomorphic Encryption, and Secure Multiparty Computation.

2.1.1 Zero Knowledge Proofs (ZKPs)

Zero Knowledge is a method by which one party can prove the validity of a statement to another party by issuing a cryptographic proof (a Zero Knowledge Proof, or ZKP) of the statement's validity. The other party can verify this statement without needing to know anything more about the statement itself (hence "zero knowledge").

It is beyond the scope of this paper to explore the technical specifications of the various types of Zero Knowledge Proofs. Our purpose here is to discuss the properties and tradeoffs of ZKPs, before applying these insights to analyzing ZKPs' computational privacy potential for blockchain applications.

In terms of computational privacy, ZKPs can be powerful, but *only if the proving party can be trusted not to leak data.* [2] Specifically, for a ZK solution to work for computational privacy, the relevant parties would have to accept the fact that the proving party sees all the data that it is writing a proof about. This requirement is trivial if the proving party is the one who owns the sensitive data in the first place, and has no incentive to leak it. In other words, this requirement is trivial if the individual user is the one issuing the ZK Proof.

However, this limitation screens off many use cases. For example, consider an application for a sealed bid auction, where each party submits a bid for an item. The application wants to privately check all the bids to only output the winning bid and complete the auction. In ZK, there is no clean way to both preserve privacy and allow for bids to take on arbitrary values. [3] If the bids can take on arbitrary values, in particular being arbitrarily close to one another, some entity must be able to directly compare all the bids, violating the condition that the bids be checked privately.

In addition to the trust assumptions of ZKPs—the need to trust prover node(s) not to leak data—ZKPs also have developer-facing limitations. While the most advanced versions of ZKPs are extremely cheap and easy to verify, it is often non-trivial to write proofs for certain types of computational logic. [4] Specifically, these advanced ZKPs are generally extremely specialized to certain

[2] See e.g. https://ethereum.org/en/zero-knowledge-proofs/

[3] Even in papers like https://courses.csail.mit.edu/6.857/2019/project/18-doNascimento-Kumari-Ganesan.pdf, the auctioneer still must see all values

[4] Even systems like Mina and Aleo limit programmers to certain non-ergonomic syntax for programming, see e.g. Mina using ternaries instead of conditionals

types of arithmetic tasks, such as range proofs and addition/subtraction, and more generic ZK methods are generally more expensive to prove. Additionally, ZK systems are generally built around data from particular sources (e.g. a specific chain). It is between difficult and impossible, depending on the ZKP implementation, to extend a particular ZKP system to verify additional data sources.

Nevertheless, ZKPs distinguish themselves from other computational privacy methods in their simultaneous low verification expense and mathematical non-falsifiability. Unlike some of the alternative methods described below, ZKPs do not rely on specialized hardware (TEEs), particularly expensive computations (FHE), or majority-honest constructions (SMPC). As we will see, these properties render ZKPs extremely promising and highly applicable to blockchains in general, but not necessarily to privacy-preserving computation for blockchain applications, in light of the aforementioned limitations.

2.1.2 Trusted Execution Environments (TEEs)

A Trusted Execution Environment, or TEE, is a secure area of a hardware processor that can execute code without revealing the inputs or computations to the larger host machine. More specifically, TEEs do two things. First, they provide easily-verifiable validity proofs that attest that a computation that was supposed to be performed over a given piece of data was, in fact, performed. Second, they attest that no other code had access to that data, by keeping it in a separate segment of memory, encrypting data as it leaves the enclave, and following specific procedures at the startup/shutdown of the unit and in the event of CPU faults.

Beyond the privacy-preserving computation functionality that TEEs offer, which keeps data private from (i.e. non-viewable by) the computing node(s), TEEs are also highly generic and quite performant, at least compared to other privacy-preserving computation methods. [5] Specifically, TEE systems like ENARX and Gramine support wide varieties of programs (either arbitrary WASM code or arbitrary linux binaries) and do so with minimal computational overhead.

However, TEEs also have potential security vulnerabilities. Specifically, since the TEE's security comes from hardware (and, in particular, hardware that the user has physical access to), a whole host of cache-level and processor-level attacks become possible on TEEs that are not possible on other private computation solutions[6]. Whereas the security of ZK systems are based on mathematical proofs, the security of TEEs is instead based on a cat-and-mouse game whereby attackers continuously find new cache/processor level flaws, which manufacturers push out updates to fix.

Nevertheless, given their generality, developer friendliness, and computational performance capabilities—as well as the crucial fact that TEEs keep

[5] See e.g. https://medium.com/@danny$_h$arnik/$impressions - of - intel - sgx - performance - 22442093595 a for some analysis of this$.

[6] See e.g. https://heartever.github.io/files/SideChannelRisks.pdf

inputs private from processing node(s)—TEEs are very promising for general privacy-preserving computation purposes.

2.1.3 Homomorphic Encryption

Homomorphic Encryption is a privacy-preserving method that uses specific mathematical techniques in order to operate on ciphertexts without converting them back into cleartexts. To put it another way, homomorphic encryption enables the performance of certain mathematical operations on encrypted data without decrypting the data, only decrypting it once the desired result is reached. These specific operations are software dependent. Some algorithms allow arbitrary computation (Fully Homomorphic Encryption, or FHE) and others allow only certain operations (Partially Homomorphic Encryption, or PHE).

Partially homomorphic encryption[7] uses relatively cheap techniques that allow for either addition or multiplication of ciphertexts, but not both. While this may seem limited, just addition/subtraction is enough to enable several basic use cases (for one, transactional privacy becomes a relatively trivial use case, as does equality testing for things like password-based authentication). However, algorithms that use partially homomorphic encryption are unable to perform fully arbitrary computation due to their only possessing partial homomorphicity.

Fully homomorphic encryption-based systems use significantly more expensive techniques to enable fully arbitrary computation over encrypted data. These techniques allow replication of any computable system/smart contract while preserving near-maximal privacy (with some exceptions due to relatively arcane cryptographic properties that FHE by definition cannot have)[8]. However, they are expensive both in computation time and data. Data are expanded by factors of 100x-1000x, and even simple computational operations requires large amounts of operational overhead. Additionally, FHE for multiparty solutions is difficult to impossible depending on an application's constraints.

Given their current performance and latency drawbacks, FHE solutions are likely too far off to offer meaningfully generic privacy-preserving computation functionality. However, as FHE development advances, these solutions could indeed mature into some of the more robust offerings given their desirable security properties.

2.1.4 Secure Multi Party Computation

Secure Multi Party Computation (SMPC or MPC) is a privacy preserving mechanism for allowing multiple parties to perform some operation over encrypted data, such that no parties learn anything besides their designated inputs and outputs (i.e. no state or data is leaked). Like FHE, there are two kinds of MPC: specific MPC (e.g. threshold signing and randomness) and generic MPC.

[7] As found in systems like Paillier Encryption

[8] https://crypto.stackexchange.com/questions/17980/can-a-homomorphic-encryption-scheme-be-made-cca2-secure

Specific/nongeneric MPC is a set of algorithms that allow for specific operations to be performed over private data without learning that private data. A classic example is threshold signatures, in which a group of individual parties together can sign a document without any of them learning the signing key. These MPC algorithms are usually relatively fast, at the cost of only working for their specific task.

In contrast, fully general MPC algorithms allow the performance of any computation that can be expressed as an arithmetic circuit (if using polynomial MPC schemes) or a binary circuit (if using Yao's garbled circuit protocol). However, like FHE, MPC pays a steep cost for that genericity. For certain forms of MPC (like the SPDZ scheme), for each time a certain common subset of computational operations is performed, every node in the network needs to communicate with every other one, which is moderate communications/network overhead. Other forms like Garbled Circuit Variants [9] have overhead more similar to FHE, where computations take up more room/are far larger per bit.

Given generic MPC's performance/latency issues, MPC-based solutions (like their FHE counterparts) are not suitable for all fully private arbitrary computation. However, these solutions could plausibly mature into readiness for such use cases, and specific algorithms like threshold signatures are already useful for certain tasks like distributed signing of documents. Additionally, for cases when multiparty security is desired and TEEs hardware risk is deemed too high, MPC is potentially a feasible alternative with the right underlying infrastructure.

2.2 Computational Privacy Infrastructure for Blockchain Applications

Having detailed some of the design tradeoffs between ZKPs, TEEs, FHE, and SMPC, we can now address the more practical question of what these technologies offer blockchain applications vis-a-vis privacy-preserving computation.

2.2.1 Analytical Framework: Correctness and Privacy

The primary additional factor to consider when assessing computational privacy solutions specifically for blockchain applications is the question of trust. Blockchains are designed to coordinate financial activity between parties without reliance on a centralized intermediary or source of truth, and the applications built on top of blockchains reflect this base assumption. Accordingly, any piece of infrastructure that proposes to provide privacy-preserving computation for blockchain applications must be trust minimized. While it is possible for public blockchain applications to outsource private computation to trusted third parties with no on-chain verification, doing so would introduce additional trust assumptions that are likely prohibitive for sufficiently decentralized protocols.

The question of trust can be concretized into a specific parameter that blockchains optimize for: **correctness**. In a word, blockchains aim to ensure result correctness without relying on a central source of truth. To do

[9] See https://dl.acm.org/doi/10.1145/22145.22178

so, blockchains utilize specific economic incentives and game theory in their consensus mechanisms to ensure that individual block producers and validators cannot easily corrupt the chain by, e.g., incorrectly reversing a valid transaction. Privacy-preserving computational infrastructure for blockchain applications that is adequately trust-minimized must also satisfy this "correctness" condition: it ought to somehow ensure the correctness of the computational outputs and the stored state of the chain.

However, there is an additional parameter that must be taken into account in order to provide privacy-preserving computation functionality over sensitive data for blockchain applications: the **privacy** of the data and state. In addition to ensuring correctness, privacy-preserving computation infrastructure must also protect the privacy of the data and state from leakage either onchain or offchain.

We now turn to the computational privacy solutions outlined above to assess their satisfaction of these criteria (correctness and privacy). We begin with Zero Knowledge systems.

2.2.2 Comparative Analysis

Zero Knowledge Proofs (ZKPs)

At the outset, it is worth noting that ZKPs have already proven to be immensely valuable to blockchain systems. ZKPs have been applied to transactional privacy—i.e. ensuring that the contents of a transaction (the sender/recipient and amount) remain private. There are promising attempts to expand on this transactional privacy foundation to enable private execution for all sorts of financial activity on public smart contract chains. Perhaps even more significantly, ZKPs have also begun to be successfully implemented as scaling solutions for Ethereum via Zero Knowledge Rollups.

And yet, our focus here is on neither transactional privacy nor scaling, but rather on computational privacy for blockchain applications. More specifically, we are looking for generic and performant privacy-preserving computation over sensitive offchain data. In terms of computational privacy for blockchain applications, ZKPs can be powerful, but only if the proving party can be trusted not to leak data.

As mentioned previously, this requirement is trivial if the proving party is the individual user. However, there are many applications that require the use of certain private data for computations over other private data. For example, privacy-preserving big data computations over healthcare records would require multiple sources of sensitive data at once; so too might an onchain privacy-preserving credit scoring contract. When it comes to these types of privacy-preserving computations beyond simple end-user attestations, one would need a central party that is running all of the computations over user data and issuing proofs onchain—which happens to be the model for most ZK Rollup constructions.

Reliance on a central prover not to leak data renders impossible a variety of potential computational applications. We noted the case of sealed-bid auctions

above as an example of an application that cannot feasibly be solved in ZK. To use an even more blockchain-facing example, let's say that Alice wants to build a private DEX factory that supports individuals using their own pricing algorithms they keep private (i.e. in case someone has some sort of "secret sauce" pricing model). For any given user transaction, in order to generate a ZKP of the price charged for that transaction, a prover would need to see both the pricing algorithm and the input user transaction—but this violates the privacy of the system.

Even more to the point, ZK based systems with central provers are only better than (non-ZK) centralized offchain computation providers in one respect: they are better at ensuring computational correctness. A ZK system such as a ZK Rollup can prove onchain that its offchain computations were done correctly, without revealing the data onchain. This is a non-trivial step forward from simply trusting offchain parties to do computation absent any type of onchain verification.

However, ZKPs do nothing to prevent a central prover from leaking data offchain. In describing the two parameters (correctness and privacy) above, we stated that data must be safe from leakage both onchain and offchain. ZK-based systems that issue onchain validity proofs have verifiable "correctness" properties, and they importantly keep data private from onchain observers. However, in these systems, there are no better guarantees against data leakage offchain than there are in existing legacy systems, which are (as recent history has proven) startlingly prone to breaches. When it comes to building privacy-preserving computation infrastructure that can handle loads of extremely sensitive data, such as social security numbers and personal health records, this limitation of ZK-based systems with centralized provers is critical.

Additionally, as mentioned above, ZKPs are still specialized to certain types of arithmetic tasks, such as range proofs and addition/subtraction, and more generic ZK methods are generally more expensive to prove. This means that, for example, applying ZKPs to more generic and complex computations (such as "big data" computations over a swath of healthcare records) proves much more difficult and inefficient, if not practically infeasible. There are promising ZK systems that have abstracted away much of the development friction for ZKPs [10], so ease of use alone is not as severe of a limitation as the offchain data leakage concerns.

The three other computational privacy techniques offer very different sets of tradeoffs to blockchain applications than ZKPs.

Trusted Execution Environments (TEEs)

TEEs, as described above, are secure areas of a hardware processor that can execute code without revealing the inputs or computations to the larger host machine. When it comes to providing privacy-preserving functionality for blockchain applications, TEEs are comparatively the most generic, performant, and seamless-to-integrate type of computational privacy infrastructure of the

[10] E.g. Aleo and Mina

four mentioned here. Unlike ZKPs, which for complex "big data" computations would require trusting a centralized prover not to leak data, TEEs keep inputs unreadable by the host node.

All of these benefits are dampened, of course, by the reliance on specialized hardware modules, as opposed to cryptography, for security. And yet, if Winston Churchill called democracy "the worst form of government, except for all the others that have been tried," then perhaps the same can be said of TEEs for privacy-preserving computation: they are not the ideal, but the alternatives are either not generic or performant enough (FHE and SMPC), or not private enough (centralized ZKR).

How can TEEs be used for privacy-preserving computation specifically for blockchain applications? The naive approach would be to have an onchain application use a single node that uses Intel SGX (a leading TEE enclave variant) to process data privately. This approach is still more private than the ZK approach, assuming no breach of the hardware integrity, since the single node would not have read access to the underlying data. However, when we return to the two desirable properties of computational privacy infrastructure (privacy and correctness), we see that reliance on a single TEE requires trust in two places even while preserving privacy. First, a blockchain application would have to trust that the code that the TEE has attested to running is actually the code that the application wants it to be running. Second, it would trust that the system state of the TEE is in fact the state that the application expects the TEE to have. On this second point, while experienced users could manually check that information every time, there is no way to enforce these checks programmatically, and moreover, some TEEs might have closed-source runtimes and system states.

To mitigate these trust assumptions, TEEs can be "brought onchain" themselves, by having a blockchain operate compute nodes in TEEs. The blockchain then continually attests, via consensus, to (a) the code that each TEE is supposed to be running and (b) to the state and data with which the TEE is supposed to run the code. TEE blockchains, as these systems can be called, offer stronger liveness and state assurances than individual offchain TEE service providers. There are multiple TEE blockchains in production today that offer consensus-controlled privacy-preserving computation. Among these blockchains, three are worth mentioning: Secret Network, Oasis, and Phala Network.[11]

TEE blockchains are a compelling solution for the problem of computational privacy for blockchain applications. However, there is one additional problem,

[11] Secret runs a cosmwasm variant on their TEEs, while Oasis has multiple different smart contract engines. Secret and Oasis have different approaches to larger-scale consensus: in Secret, the consensus layer is also the TEE (computation) layer, while in Oasis, the consensus layer is separated from the computation layers (called Paratimes). However, the primary difference between the two networks is in their stage of development: Secret is live on mainnet with a private computation product, while Oasis is only on testnet with their general compute (their Cipher and Sapphire paratimes). Phala Network is another Polkadot Substrate-based TEE blockchain worth mentioning, which focuses on having off-chain TEE nodes perform computations verified on chain. They are on mainnet but have yet to find significant market fixation.

beyond the security questions we have already mentioned. ZK systems are natively interoperable with public smart contract blockchains. ZKPs generated offchain—either by an individual user or by a central prover—can be easily verified onchain without any additional interoperability infrastructure. TEE blockchains, however, are natively separate and disconnected from smart contract blockchains like Ethereum. Until now, developers of these systems have chosen to bite the bullet and embrace a siloed approach: these TEE chains have mostly been developed as stand-alone, "monolithic" smart contract blockchains designed to do everything that public smart contract blockchains can do, and also do it privately using TEEs.

However, this monolithic approach to blockchain development will not, we are convinced, serve as an adequate solution. Not only is it the case that public smart contract blockchains are leagues ahead of these nascent TEE chains in ecosystem development and mainstream adoption, but it is also a brute fact that TEE chains are (a) more complex for developers to build on, and (b) limited in their scaling capacity, because they have to do everything that public chains can do except privately in TEEs. Even functions that do not necessarily need to be private—for example, a flash loan—are executed privately and thus more computationally expensively than on public smart contract blockchains.

Aside from the general security tradeoffs of TEEs, the main problem with TEE *chains* is that they are siloed. They have the capacity to perform trust-minimzed and privacy-preserving arbitrary computation for public blockchain applications, but in their current state they are relegated to performing only internal, native computations. This is a problem to which we will return in greater depth below, since this is fundamentally the animating problem that Snakepath aims to solve.

Fully Homomorphic Encryption (FHE) and Secure Multiparty Computation (SMPC)

As for Fully Homomorphic Encryption (FHE) and Secure Multi Party Computation (SMPC), the remaining two computational privacy methods described above, both of these offer stronger privacy guarantees than TEEs (not to mention ZK systems with central provers). However, given the performance and latency limitations for both methods when applied to arbitrary private computation, neither constitutes a fully viable solution to the need for generic and performant privacy-preserving computation for blockchain applications. FHE is also limited by its primarily single party nature—while there exist multi-key FHE protocols[12], they end up having similar security properties to MPC-like systems.

With that said, there are still some early projects that utilize one or both of these two privacy techniques in order to offer computational privacy to blockchain applications. Zama is planning to use FHE to provide privacy to their blockchain, while Aleph Zero is planning to use MPC to provide privacy to theirs. On the less generic side, projects like Unbound (acquired by Coinbase)

[12]https://eprint.iacr.org/2020/180

and Threshold are using MPC for specific use cases, and projects like Dero are using additive homomorphism to provide limited privacy.

Generic FHE and SMPC will continue to improve in performance. Arbitrary computational readiness for blockchain applications is likely a matter of when, not if, for these technologies.

Bottom Line

Zooming out, each of the computational privacy methods besides ZKPs is not natively interoperable with public smart contract blockchains. While TEEs, FHE, and SMPC all have an advantage over central-prover ZK systems in that they do not trust the computational node(s) not to leak data, all of these systems—including the TEE blockchains currently in production—do not directly provide privacy-preserving computational services to public blockchain applications. Put differently, once we move beyond the world of user-generated ZKPs and central-prover ZK Rollups, we are left with computational privacy technology that may be viable along the lines of the desired properties stated above (i.e. correctness and privacy). Yet these technologies, despite their promise, require additional interoperability infrastructure in order to serve public blockchain applications.

Snakepath, the protocol we outline in the rest of this paper, is our solution to the lack of generic interoperability to bring computational privacy to public blockchain applications.

3 The Snakepath Protocol

The previous section established that there is viable infrastructure for privacy-preserving computation over sensitive data for blockchain applications. However, this infrastructure is by and large siloed away from the majority of blockchain applications and their users, which live on public smart contract blockchains like Ethereum that do not natively have computational privacy functionality.

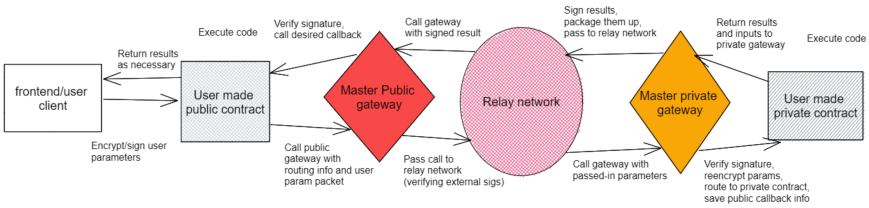
In light of this fact, the objective at hand becomes one of interoperability: namely, between public smart contract blockchains and trust-minimized computational privacy infrastructure.

Our proposed solution to this challenge is Snakepath, a protocol for secure and efficient inter-chain communication. Snakepath enables public chains to call arbitrary private functions while preserving the privacy of the inputs and validity of the outputs. In this section, we will outline the high-level architecture of the protocol and review a list of desired security properties, before diving into a step-by-step overview of Snakepath and some of the key details on various parts of the architecture.

3.1 Core Architecture of Snakepath

We begin here with a high level summary of the network and a list of desired security properties, before diving into a step-by-step overview of the protocol

and some important details on various parts of the architecture.



3.1.1 Snakepath at a high level

Snakepath is a protocol for lightweight, secure, privacy preserving message-passing between chains. More technically, Snakepath is a message passing system for non-malleable, trustless interchain message passing.

Snakepath consists of two core primitives: relayer(s) and gateways.

Relayers watch for messages on one chain and then call another chain with that message. Because Snakepath is not built to handle crosschain value transfer or generic crosschain messaging, the network architecture around the relayers is substantially different from its public peers like Axelar, Layer Zero, and Synapse. For Snakepath, the primary security-related concern is liveness, not resistance to node collusion or censorship. The relayer is not tasked with relaying funds/tokens from one chain to another, but rather with watching for encrypted messages that need to be passed. The relayer does not have access to encryption keys that can decrypt these messages or signing keys to generate new valid ones, meaning that the relayer cannot compromise the data in security or correctness. The most a relayer can do is to not transmit the data as requested, which would cause liveness problems for the network, but would not compromise user funds, smart contract applications, or sensitive data. This also makes the requirements for running a relayer significantly lower: a relayer needs only to be able to watch for transactions on host chains and create transactions on destination chains based on those host chain transactions.[13]

Gateways are the onchain smart contracts that handle the broadcasting, receipt, packaging, and verification of messages. Gateways are also the mechanism by which application developers will interface with the Snakepath protocol.

These gateways are relatively simple and generic, primarily consisting of three capabilities. First, gateways can handle signature operations. For gateways on public blockchains, that means verification; for those on privacy-preserving blockchains, that means verification as well as signing. Second, gateways handle serdes operations: converting contract inputs into correctly formatted packets and vice versa. Third, gateways on privacy-preserving chains can re-encrypt their inputs under the key of the final compute contract.

[13] Note that this design requires that relayers have sufficient gas to pay for these created transactions. Relay fees should cover these costs.

The basic and generic nature of Snakepath gateway contracts has two primary benefits. First, gateways do not need to be modified to handle different use cases. Second, gateway contracts are simple both to code and verify. The combination of these two qualities means that Snakepath is easy to scale to arbitrary chains. For example, to expand to chain X, one simply needs to build a gateway for X, and augment the relayer software to be able to read and write to the X gateway.

The design of Snakepath is purposefully lightweight when compared to other natively- and externally-verified cross-chain interoperability protocols. However, Snakepath does have one significant attack vector that does not really exist for the other cross-chain interoperability protocols: fraudulent "gateways." Specifically, in our protocol, each gateway contract needs to maintain a list of gateway contracts/signing keys on other networks that it default-trusts. If someone were to build a false gateway on some well-used network and get other gateway contracts to accept that false gateway, they could read/misprocess arbitrary packets. Solving this problem requires a reliable mechanism for key distribution/gateway identity distribution, which we will elaborate on below.

The following sections expand on the protocol in greater depth, starting with the desired security properties of the TNLS primitive, on which the Snakepath network is built.

3.1.2 Desired Security Properties

TNLS, and thus Snakepath, is designed to achieve the following security properties:

- Result accuracy:

  A public contract cannot be convinced to accept a result that did not come from the private contract it called or that came from inputs that were not the ones it passed in.

- Parameter accuracy and security:

  The parameters for a private computation cannot be modified or read in transit.

- Address association security:

  It is impossible to claim to have created a computation from an address not controlled by someone who knew the encrypted inputs.

3.1.3 Key Distribution Scheme/Protocol Neutrality

As mentioned above, Snakepath requires some mechanism for reliable key distribution for the on-chain gateways. In order to do this in a decentralized way, we propose using a threshold signature-based key management system, wherein the on-chain gateways will only recognize new gateways if their public signing keys are signed by the threshold signature key.

In effect, we use a network of relayers as a decentralized certificate authority, to remove the lingering point of failure in traditional CAs which is the compromise of the root cert granter (see, e.g., Superfish). Additionally, this relayer network can trustlessly be incentivized to do message passing if running a single relayer proves not to scale, since our protocol prevents modification or reads in transit. The only negative impact a malicious relayer could have is deciding not to transmit messages, which is equivalent to their nonexistence.

3.1.4 Step-by-step Protocol Overview

Having detailed the desired security properties and key distribution scheme, we now provide a step-by-step technical overview of the TNLS primitive that underlies Snakepath.

```
User U encrypts (with the private chain gateway, Gate_priv's,
    public key) and signs (with their own private signing key)
    a packet Pack containing:

        The data U wants to transmit
        Where U consents for that data to be routed
        U's verification key/public chain address

and sends Pack to the public chain contract, C_pub, U is interacting with

C_pub sends Pack to the public chain gateway, Gate_pub,
    with routing information on the private chain contract C_priv
    that needs to process it, and the source user U.

Gate_pub verifies the signature on Pack came from U and broadcasts the packet,
    source user and routing info to our relayers, R,
    along with a generated task ID T.

R uses that routing info to send Pack
    and U's verification key to the desired private gateway, Gate_Priv

Gate_Priv verifies the signature on Pack and then decrypts it,
    verifying that U matches Pack's stored address and
    that the routing info is in Pack's stored consent packet.

Gate_Priv then privately calls the private compute contract,
    C_priv, with the transmitted data.

C_Priv computes some information over that data and
    calls Gate_priv with the tuple of  (results, pack, T).

Gate_priv signs the tuple of (Results, Pack, T)
```

```
    and broadcasts that tuple back to R

R sends that signed tuple back to Gate_pub

Gate_pub verifies the signature with Gate_priv's verification key
    and that Pack matches its stored Pack,
    and then calls C_pub with the results.

C_pub can now use those results, knowing Results verifiably came from processing Pack.
```

The protocol has the following primary gas costs:

```
Gate_pub needs to verify the signature of Pack matches U's address.

Gate_priv needs to verify the signature of Pack matches U's address,
decrypt Pack,
verify the routing info matches the consented to routing info,
and deserialize Pack.

Gate_priv needs to sign the result tuple and serialize it.

Gate_pub needs to verify the result tuple signature and that inputs match.
```

Now that we've discussed the overall protocol, we can discuss the gateway structure.

The public gateways will have the following entrypoints:

- Store new public key:

  This entrypoint takes a pair of route and public key, signed by a master key (held potentially as a threshold signature-based key, as described above) and saves them for future verification.

- Broadcast user message:

  This entrypoint takes in a user packet, callback, and routing info, verifies the signature of the user packet, generates a task ID, saves the callback, and broadcasts the packet, task ID, and routing info as an event.

- Receive interchain message:

  This entrypoint takes an interchain message (result, packet, task ID, source network) from the relayers, verifies that (a) the routing info is as expected, (b) the signature matches the saved key for the source network, and (c) the packet matches the expected task ID, and then calls its saved callback with the result.

The private gateways will have the following entrypoints:

- Receive interchain message:

  This entrypoint takes an interchain message (packet, task ID, user address) from the relayers, verifies the user signature, decrypts the packet, verifies the internal verification key matches the user address, verifies the routing info with the packet matches the internally stored routing info, saves the task ID, packet and decrypted parameters, and calls the contract/handle in the routing info with the decrypted parameters and task ID.

- Broadcast contract message:

  This entrypoint takes in the result of a contract and the task ID/parameters it was called with, verifies that the task ID, parameters, and contract match what was expected, then signs and broadcasts the tuple of [Result, packet, task ID].

4 Comparing Snakepath to Existing Interoperability Infrastructure

Now that we have overviewed Snakepath, it is worth briefly analyzing the existing interoperability landscape to see where Snakepath fits in. Specifically, we will discuss how Snakepath fares against existing interoperability protocols with respect to our target objective: namely, efficiently and securely facilitating privacy-preserving computation for applications on public blockchains.

4.1 Crosschain Interoperability

There are a variety of cross-chain interoperability protocols in use today, among them Synapse, Layer Zero, Axelar, Wormhole, Celer, Cosmos IBC, and Connext. It is beyond the scope of this paper to examine their respective architectures and security assumptions, though much has been written on this subject. We only mention these protocols here because, like Snakepath, they broadly facilitate inter-blockchain communication.

However, unlike Snakepath, the aforementioned protocols all focus on cross-chain value (i.e. token) transfer and cross-chain message passing. The purpose of the latter (cross-chain message passing) is to enable the building of natively cross-chain smart contract applications—e.g., a DeFi lending protocol that allows users to take out a loan on Chain X using collateral on Chain Y without needing to transfer that collateral to Chain X.

The objective at hand is entirely different: enable public chain applications to access privacy-preserving computation on other chains. Unlike these existing interoperability protocols, Snakepath does not interoperate between public chains, since TNLS assumes that at least one side of the bridge can securely encrypt and sign messages, which public blockchains like Ethereum do not do. As a result, Snakepath is not designed for cross-chain token transfer, nor can it be used to build crosschain smart contract applications on public chains alone.

Instead, Snakepath enables applications on public chains to call arbitrary private functions (over private data) on privacy-preserving chains. Almost all other interoperability protocols do not support this purpose, since they all transfer data in clear text and cannot easily be adjusted to facilitate encrypted message passing. Moreover, due to their constraints and security assumptions, these other interoperability protocols either need multi-sig- or lite-client-based verification, which greatly increases their gas overhead for simple message passing.

Specifically, due to our desired security properties and the upsides enabled by on-chain private signing and verification keys, we can verify message integrity *without* needing to verify the entire state of the chain as part of bridging a packet over. This entirely eliminates the need for the higher-overhead parts of standard bridging protocols, but does so in a way that requires a ground-up rewrite of those protocols to eliminate the expensive steps of multisig or state proof packaging/verification.

The as-yet unreleased Secret IBC (and more generally the set of IBC and multisig bridges Secret Network maintains) is the only cross-chain bridge (that we know of) that supports encrypted message passing. Secret IBC is planned to enable privacy-preserving value bridging and some amount of low-level data bridging, but there are efficiency concerns with this model. Specifically, while Secret IBC allows encrypted message passing, it does so as a level above standard Cosmos IBC, with encryption managed on the client side. As a result, Secret IBC is subject to the same lite client efficiency constraints.

Seperately, Secret's multisig bridges have the same efficiency issues as standard multisig implementations, since they purely add encryption as an additional step in the process. Additionally, these systems are limited only to connecting the Secret Network to public blockchains—it is not a neutral bridge that can connect public blockchains to any privacy-preserving computation chain.

4.2 DECO

While not a cross-chain interoperability network, DECO is worth mentioning as it broadly falls under the interoperability umbrella—except for bridging web2 data privately to web3 (i.e. onchain) without any additional privacy-preserving computation functionality. DECO (short for decentralized oracle) is a cryptographic protocol designed by Dr. Ari Juels, among others, for proving that a particular piece of information came from a particular web2 server over TLS. Additionally, the protocol can be used to prove certain statements about that information in zero knowledge.

DECO, though not yet in production, could be useful for an application that wishes to bring information from a web2 server onchain—so long as it is comfortable with one party performing all of the computation and proving statements about the result in zero knowledge. For some use cases, this scheme may be sufficient. For a KYC check, for example, a person could verify that they are who they say they are by proving in zero knowledge that a statement of identity (e.g. "This Person is John Doe") came from a TLS session with a government body or otherwise-trusted identity verification source.

There are two main differences between Snakepath and DECO. First, DECO is designed for interfacing with servers that already have TLS up and running. In other words, DCO provides a way to bring proofs of TLS sessions onchain. In contrast, Snakepath is a protocol for secure interchain communication that also extends to web2 servers. In effect, Snakepath is a lower level primitive than DECO. Second, DECO provides privacy-preserving functionality via the mechanism of zero knowledge, while Snakepath is more generic and extends to other computational privacy solutions such as TEEs, SMPC, and FHE.

DECO is a protocol purpose-built for one thing and one thing only: getting data and ZK statements about that data from web2 to web3. For that task, it is quite efficient. However, for the separate task of "one blockchain privately performs a computation and sends the result to another blockchain (or web2 server)," DECO does not offer a solution, while Snakepath does.

5 Potential Applications of Snakepath

Snakepath itself is not a privacy-preserving computation network; the protocol itself does not actually facilitate any types of computation or execute any transactions. Snakepath is, in this sense, a simple, lightweight standard for encrypted message passing between (public and privacy-preserving) chains.

However, Snakepath is a necessary piece of infrastructure for public blockchain applications that require privacy-preserving computation functionality. In this sense, Snakepath does unlock a whole suite of new potential functionality for public blockchains. This functionality can be split into two categories: (1) applications that need to privately compute over offchain sensitive data, and (2) applications that require privacy-preserving computations over onchain data that keep inputs private from computing nodes.

As far as offchain sensitive data goes, there are a variety of blockchain applications that can be built if they have access to privacy-preserving computation functionality. Within the realm of financial data, Decentralized Financial (DeFi) applications on public chains could operate privacy-preserving contracts on separate chains via Snakepath. These contracts could privately compute credit scores over personal user financial data, such as income statements or asset reports. Now, as things currently stand, DeFi applications could of course trust users or credit providers to "attest" onchain to their credit score (even using a ZK proof for private verification). However, enabling DeFi applications to privately run their own computations, as opposed to simply accepting black box computations done by offchain entities, reduces the trust assumptions that DeFi applications need to make.

To play this example out a bit further: If a lending protocol on Ethereum today were to attempt to determine users' creditworthiness, it would have to do one of two things. Either it would force users to upload sensitive financial history and identifying documents onchain, or it would need to fully trust an offchain credit bureau. The first option is obviously a non-starter. The second option is a possibility but has various problems: How does the lending protocol

verify that the score is computed correctly? How can the user be sure that the credit bureau doesn't share his financial history with other third parties? What if the hidden credit algorithm itself has built in (ethnic, racial, religious) biases? Privacy-preserving onchain computation can materially reduce the trust assumptions that blockchain applications make when processing or verifying offchain data.

To be sure, the availability of trust-minimized privacy-preserving computation does not solve the "oracle problem": when it comes to bringing offchain data onchain, there will always be some sort of source of truth that the chain needs to accept. Even if an onchain application can perform the necessary computations more transparently (while privately), it must still accept raw data from some offchain source.

However, Snakepath actually points the way toward a provisional "solution" to the oracle problem for sensitive data: applications can, via Snakepath, compute over multiple offchain data sources at once, minimizing reliance on single entities and building a sort of pseudo-consensus mechanism for offchain sensitive data. This type of multi-source computational infrastructure is even more reason why Snakepath is necessary, since simple ZK constructions with end-user attestations would be more difficult to build in a multi-source fashion. With Snakepath, applications on public blockchains can effectively create multi-source computational oracles, with computational logic living on privacy-preserving chains.

Beyond financial data, there are other types of sensitive data that public blockchain applications could process using Snakepath. One important type to mention here is healthcare data. As much as five years ago, entrepreneurs and investors touted blockchain's potential in medical technology. Those lofty promises have gone largely unfulfilled, and "health-tech on the blockchain" now (justifiably) attracts the ire of most crypto natives. Nevertheless, with robust and trust-minimized computational privacy infrastructure that can be seamlessly connected to public blockchains via Snakepath, it is now more feasible to incorporate sensitive healthcare data into onchain applications than has been until now. For example, open, permissionless data marketplaces on public blockchains like Ethereum could utilize privacy-preserving computation networks via Snakepath to enable users to "stake" and sell their genetic or clinical data to potential buyers such as healthcare AI startups or drug discovery companies. These buyers would then only have access to the data from within the confines of the privacy-preserving systems to which the data marketplace is connected.

Beyond enabling applications on public blockchains like Ethereum to securely process sensitive offchain data, Snakepath also unlocks computational functionality for onchain data that requires fully private computation. Earlier we mentioned sealed-bid auctions as a classic example of an application that cannot feasibly be built in zero knowledge. Developers could build applications like this one—i.e. one that mandates input privacy from the computing node(s)—directly on privacy-preserving blockchains, like TEE chains. However, these applications would not be easily accessible to, or composable with, public

blockchain applications and their liquidity. Snakepath enables public blockchain applications to take advantage of privacy-preserving computation features on these other chains, such that, for example, a sealed-bid auction application on Ethereum could outsource private bid processing to the Secret Network or another TEE chain. Nevertheless, this functionality (privacy-preserving execution) as opposed to privacy-preserving computation over offchain sensitive data is less urgent overall because of the general availability of Zero Knowledge solutions that act as a "VPN" for DeFi transactions.

Our ultimate goal, however, beyond any of the functionality described above, is to lay a foundation that is open, transparent, and credibly neutral. If successful, Snakepath will enable other developers and entrepreneurs to build natively-interoperable, private computation-based blockchain applications that we, the authors, cannot even conceive of.