Anoma Research Topics | REPORT

Exploring Cryptographic Approaches to Enhance Privacy in Intent Solving

Yulia Khalniyazova[a]

[a]Heliax AG

* *E-Mail: yulia@heliax.dev*

Abstract

In this report, we explore cryptographic strategies that could be used to enhance privacy during the phase of solving intents in Anoma. We consider both well-known solutions like trusted execution environment, multiparty computations and homomorphic encryption and more exotic primitives like functional encryption, witness encryption, searchable encryption and collaborative SNARKs. We identify the limiting factors for each primitive and provide a summary of the results.

Keywords: Collaborative SNARKs; Functional encryption; Homomorphic encryption; Intents; Multiparty computation; Private solving; Searchable encryption; Trusted execution environment; Witness encryption;

(Received September 5, 2021; Revised September 15, 2021 ; Published: October 5, 2023)

1. Introduction

In the intent-centric terminology of Anoma, *solvers* are the actors who receive user intents, turn them into transactions by matching intents together, and send the resulting transactions to the settlement engines. The phase of finding counterparties and matching intents together is referred to as a counterparty discovery phase, and the rest is referred to as a settlement phase.

1.1. Settlement privacy vs counterparty discovery privacy. . Anoma is designed to provide privacy on the settlement layer, meaning that a transaction settled on the blockchain doesn't reveal any information about the state transition unless the user involved in this transaction permits it. The user can decide how much and to whom to reveal the information about the state transition. Settlement privacy has been discussed for years, and some fantastic solutions have been developed to address the issue of programmable settlement privacy (Hopwood et al., 2022; Xiong et al., 2022). In Anoma, settlement privacy is achieved with the help of recursive zero-knowledge proofs and encryption.

Counterparty discovery privacy refers to the user's ability to keep their intents private during the solving phase. Due to the lack of existing generalised intent-centric protocols, privacy has not been discussed in this context at all. At the same time, counterparty discovery privacy requires approaches different from settlement privacy, as it faces different issues: how can a solver match your intent without seeing it?

The current approach to solving intents allows solvers to have access to plaintext intents. Users have to trust that the solvers do not take advantage of the users from knowing their intents. One of the common ways to reduce the need for absolute trust is to apply cryptographic tools. This report examines various cryptographic techniques that could be beneficial in reducing the amount of data that must be given to solvers to match intents and produce valid transactions.

2. Tools

In this section, we introduce various cryptographic techniques we have considered in the context of private solving.

2.1. Trusted execution environment. Trusted execution environment (TEE) is an isolated environment of a hardware computing device that allows to perform computations securely, separately from unprotected computations. The data is transferred to and from TEE in an encrypted form, and only decrypted inside the TEE. One of the most well-known examples of TEE are Intel SGX and Apple iOS Secure Enclave.

TEE has some advantages (e.g., performance, latency) compared to other cryptographic solutions, like multiparty computations and homomorphic encryption, which are often considered in similar contexts, and is used by multiple projects to provide privacy (Annessi, 2023; Network).

2.2. Multiparty computations. Multiparty computations (MPC) allow multiple parties to compute a public function together. Each party holds a secret share that they use to compute their public shares, which can be revealed to other parties. Parties jointly compute the output of the desired function by communicating and performing computations on public shares.

MPC frameworks are a tool that is often used to write protocols computing arbitrary functions in the MPC settings. Such frameworks provide a language to define secret shares and the function to compute. The program is compiled to a circuit that will run by the protocol participants.

2.3. Homomorphic encryption. Homomorphic encryption (HE) allows performing operations on encrypted data without decrypting the data. This can be represented as:

$$f(E(x)) = E(f(x)),$$

where

- E denotes the encryption function
- f represents the homomorphic operation being performed.
- $f(E(x))$ signifies performing the homomorphic operation on the encrypted data.
- $E(f(x))$ indicates the encryption of the result of the homomorphic operation performed on the original data.

There are two operations that are usually discussed in this context: addition and multiplication. If the homomorphic encryption scheme supports both operations computed unlimited amount of times, the encryption scheme is called fully homomorphic. If it supports either addition or multiplication on encrypted data, it is called partially homomorphic. Fully homomorphic encryption allows us to perform arbitrary computations on encrypted data. It is a powerful cryptographic primitive that is being actively researched.

2.4. Witness encryption. In witness encryption, any NP problem can serve as a public key, and a solution to the problem, called witness, acts as the corresponding private key (Garg et al., 2013). Witness encryption is a non-interactive primitive; it is relatively new (the first paper introducing it was published in 2013) and does not have well-known applications or practical implementations yet.

2.5. Functional encryption. Functional encryption (FE) is a primitive that allows different parties to obtain outputs of different functions computed on the same plaintext instead of seeing the plaintext itself as a result of decryption (Boneh et al., 2010). One of the applications of functional encryption is access control, where parties with different access levels have different visibility of the data. It is an active area of research and is not yet widely adopted.

2.6. Searchable encryption. Searchable encryption (SE) is a cryptographic technique that enables searching on encrypted data using (encrypted) keywords (Boneh et al., 2004). In comparison to homomorphic encryption, searchable encryption focuses on the specific problem of encrypted search and is optimised to solve this problem.

2.7. Collaborative SNARKs. Collaborative SNARKs allow multiple parties to cooperate to create a joint proof of knowledge (Ozdemir and Boneh, 2021). Typically, a SNARK prover holds a witness and creates a proof stating that the witness satisfies certain constraints. Co-SNARKs allow multiple provers, each of which holds only a share of a witness, to produce a single proof about the witness without explicitly reconstructing it. Co-SNARKs are constructed by applying MPC techniques to SNARKs.

3. Application and Analysis

In this section, we consider how the cryptographic techniques introduced above can be applied to increase user privacy in the solving phase.

The goal is to let users keep their intents secret, but still allow them to work with solvers to produce shielded transactions that satisfy the intents of all involved users. Instead of sending raw intents, users would share some non-sensitive data with solvers (e.g., encrypted intents). This data should be enough for the solvers to produce shielded transactions that satisfy the intents. The solver's algorithm might be public or private.

We are considering the situation where a solver is a mediator of the user interaction but might also be a user themselves. The scenario where users solve their intents themselves without an explicit mediator does not seem to significantly affect the results of the analysis.

3.1. Trusted execution environment. The usage of TEEs in the defined setting can be described as follows.

- Users encrypt their intents and send them to TEE. Only TEE can decrypt the intents.
- A solver sends their solving strategy to TEE.
- TEE decrypts the intents and runs the solving algorithm on them, outputting a shielded transaction that satisfies the intents.

This way users do not need to show their intents to the solver and the solver's strategy is independent of the intents being processed. Note that unlike the approaches where TEE is used for settlement privacy, in our context TEE wouldn't store any data long-term.

The solver's strategy must be proven not to violate any rules and not take advantage of the users (beyond expected). If the strategy is public, it can be reviewed publicly and proven to be the same as the one sent to TEE. If the strategy is private, zero-knowledge proofs might be considered as a way to prove the compliance with the defined policy.

A downside of using TEE is that it is infamous for being broken regularly, multiple hardware attacks have been discovered over the years (van Schaik et al., 2022; Moghimi, 2023), along with other threats like side-channel attacks and covert channels (Annessi, 2023), when untrusted inputs might alter the program execution and leak private information. For these reasons, many consider TEE only helpful for defence in-depth but not as the main security component to rely on.

In the case when TEE is used as a primary tool to provide privacy, the exact technology (e.g., Intel SGX) has to be thoroughly examined to make sure the implementation is not vulnerable in the presence of various attacks.

3.2. Multiparty computation. In the defined context, users private shares would be intents. The result of the execution of the MPC protocol is a shielded transaction that satisfies the intents of all users involved in a transaction.

One of the disadvantages of using MPC protocols is that when solving intents, we cannot process one intent at a time (because intents of multiple users are matched together), and because it cannot be known in advance what intents will be matched, all users that want to have a chance for their intents to be satisfied would have to participate in the MPC protocol. For the same reason, some intents submitted to the MPC protocol will not even be included in the produced transaction.

Many of the MPC protocols are limited in the amount parties they can support, and the protocols that support a large enough number or even arbitrary number of parties have the communication cost going up with the amount of participants.

To address this issue, a less generalised version of MPC solving could be used, where intents of a smaller number of parties are attempted to be matched. Having fewer parties positively affects the efficiency of the protocol but leaks more information (e.g., if attempting to match intents of two parties together doesn't lead to a transaction creation, the solver learns that the intents don't match), on the other hand increasing the number of parties negatively affects the efficiency but allows to have less transparent results (e.g., having ten intents as input instead of just two will more likely result in a match of at least two parties, and and it will not be clear whose intents got matched).

Another thing to take into account is the security model of the MPC protocol in use. Is it honest majority or dishonest majority? Does it support aborts, does it work in the malicious adversary settings? All of these factors directly affect the number of rounds, the amount of data transmitted, and the computation complexity. Generally, the weaker the assumptions are, the more expensive it will be to run the protocol.

Another aspect to pay attention to is the protocol complexity. Having a fairly non-trivial protocol that involves communication of large amount of parties with evaluation of the solving strategy, computing zero-knowledge proofs to produce shielded transactions as an output might sum up into an expensive protocol to run.

The protocol complexity can be reduced by evaluating only simple solver strategies or only matching intents that do not require additional partial transactions to be created by the MPC solver. Such transactions do not necessarily require creation of zero-knowledge proofs, which significantly reduces the complexity of transaction creation.

Combining the cost of the number of participants, the security assumptions, and the circuit size, we end up having a protocol that might be quite expensive to run.

To have more concrete estimations, we wrote a proof-of-concept (Khalniyazova, 2023) for two simple strategies using MP-SPDZ framework (Keller). Both strategies are defined for a two-party intent-matching protocol. The first strategy takes users' intents and checks that they match exactly. Such strategy does not require creating new partial transactions provided the initial partial transactions from users, but it does require creating a transaction (which involves signature creation) and is quite limited by the requirement of exact match (including amounts). The second strategy takes users' intents, checks that the requested token types and desired ratios are compatible, and determines the amounts to be sent by each user. Such strategy would require creation of additional partial transactions, which would require calling Taiga and creating ZK proofs as a part of the MPC protocol.

The first strategy takes 14 seconds to run and requires about 863 MB data transferred in a protocol with 563 rounds. The second strategy takes 18 seconds to run and requires about 1083 MB data transferred per party in a protocol with 715 rounds. Cryptographic computations within the MPC protocols for the defined strategies would affect the protocol costs the most, but they were not included in the proof-of-concept due to the complexity of the required computations and limitations of the used framework.

3.3. Homomorphic encryption. Using homomorphic encryption, each user would encrypt their intents, send them to the solver, and the solver would match the encrypted intents to produce a shielded transaction.

It is known that having the ability to compare encrypted data to a known constant makes the homomorphic encryption system lose its security properties (Ronald L. Rivest, 1978), and in our setting, the solver has such ability, being a user themselves. Having an encrypted intent, the solver would produce their own encrypted intent, and see if the two intents match. If they do not, the solver would modify their intent and try again, repeating this process until the intents match, recovering the user's encrypted intent.

Ensuring that the solver cannot be a user in this context might help to mitigate the issue, however, it might be potentially impossible for two reasons:

- the solver could always obtain a new, not associated with the solver, identity (e.g. creating a new account, or asking other users to help),
- doing so on the protocol level (if possible) would prevent the solver from ever participating in a protocol as a user, which conflicts with the ideas of censorship resistance.

Another concern is the performance of fully homomorphic encryption schemes. Although it is being actively developed, it might not be good enough to instantiate such a protocol without a noticeable loss in performance.

3.4. Functional encryption. As we are trying to protect the intents, naturally, they would be the data to encrypt using functional encryption. What will the decryption function(s) be? The expected result of the computation of private solving is a shielded transaction, meaning that the decryption function's output would have to be a shielded transaction.

In (Boneh et al., 2010), a functionality F is defined as a function $F : K \times X \to \{0, 1\}$ described as a *deterministic Turing machine*, where K is the key space and X is the plaintext space. The decryption function is defined as $\text{dec}(k, c) = y$, where k represents the secret key, c denotes the encrypted message, and y is the function's output.

Each shielded transaction contains multiple Halo2 proofs all of which would have to be computed in the decryption function. Each proof involves generating some random parameters, e.g., blinding. Because a decryption function is described as a deterministic Turing machine, it is not possible to generate randomness within the decryption function. If we are considering providing the randomness from outside the decryption function, who would produce it? The solver's randomness is not to rely on. Provided by the user, the randomness would have to be fixed per intent. An alternative approach to generate randomness could be the use a cryptographic randomness source, e.g., threshold BLS signatures.

Another question is how to combine intents together in the decryption function. It would have to take other intents as inputs, decrypt them (the decryption keys would also have to be provided in the input), and perform the rest of the computations. Is it possible to make sure the solver doesn't decrypt the input intents? The level of complexity and feasibility of this scheme are also questioned.

Besides that, functional encryption suffers from the same problem as homomorphic encryption: how to prevent solvers from testing encrypted intents?

Overall, functional encryption seems to be an interesting primitive to consider in combination with other techniques, however, it does not seem to be possible to use it on its own. It is also quite new and at the moment, there are no stable schemes that offer the required security guarantees and expected performance.

3.5. Witness encryption. Witness encryption allows to decrypt a message if the party that is willing to decrypt it knows a solution of some NP problem, otherwise witness encryption schemes act like vanilla encryption schemes. That means witness encryption does not allow operating on encrypted data and decryption, unlike in the case of functional encryption, fully reveals the plaintext. Matching intents encrypted with witness encryption would require decrypting them, which does not fulfill our requirements.

3.6. Searchable encryption. Searchable encryption seems to be a good option when the data we want to hide is not the same data we want to use as keywords. Unfortunately, it is not the case with intents.

Having intents encrypted with searchable encryption, a solver would look through intents and scan them for the defined keywords. However, a malicious solver could always scan intents for all available keywords, essentially recovering the intent plaintexts. Considering that the solver can always be a user, limiting solver access to keywords would not help here: if users want their intents to be solved, they need to make sure the intent can be searched through by other users.

3.7. Collaborative SNARKs. Collaborative SNARKs allow us to produce a joint proof from a shared witness, but do not allow us to perform computations, which is the result we want to produce. Because of that, collaborative SNARKs don't seem to be applicable in our context.

4. Conclusion

The table below contains the summary of the cryptographic approaches we have considered in this report. It is clear that there is no simple answer to the question of private solving; none of the cryptographic techniques appear to be sufficient on their own.

Primitive	Downsides
TEE	Hardware vulnerabilities, side channels, covert channels
MPC	Scalability. Number of parties, circuit size, security assumptions
HE	Comparing to a known constant allows the recovery of a message
FE	Hard to obtain randomness. Complexity and feasibility are questioned
WE	Does not allow performing private computations
SE	Search keyword coincides with the data being hidden
Co-SNARKs	Do not allow performing private computations

Combining various cryptographic and non-cryptographic techniques might allow for better practical privacy guarantees compared to the current state. With cryptographic approaches, there are two aspects to pay attention to:

- Solvers are users. Solvers can see as much as users can, so the approach used to keep the intents private should work against both users and solvers.
- Intents are matched together, and none of the participants are known in advance. Therefore, the privacy-preserving protocol for solving intents must not rely on static participants, it should scale well or be independent of the amount of parties.

4.1. Non-cryptographic mitigations. There are certain non-cryptographic techniques that can be considered to achieve better privacy or make the use of the cryptographic tools more sensible, e.g., producing an output transaction regardless of the result of the computation and solving intents in larger batches, or using stronger security models like honest majority setting for MPC protocols.

Another approach to consider is partially interactive counterparty discovery, in which a user uses a less precise version of their intent (perhaps, with some noise added to it) to find the solvers who could potentially match it. Such approach doesn't allow to hide the intent from the solver who produces the final transaction but helps to reduce the total number of parties who see the intent as opposed to the case where the user broadcasts their intent to the network of solvers.

4.2. Future work. Although a general solution to achieve privacy in counterparty discovery phase has not been found, less general, case-specific solutions for private solving might be a promising direction for future work. After all, not having a general solution is fine, as long as we have case-specific solutions for each case.

References

Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash Protocol Specification. 2022. URL https://github.com/zcash/zips/blob/main/protocol/nu5.pdf. (cit. on p. 1.)

Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. Veri-zexe: Decentralized private computation with universal setup. Cryptology ePrint Archive, Paper 2022/802, 2022. URL https://eprint.iacr.org/2022/802. https://eprint.iacr.org/2022/802. (cit. on p. 1.)

Robert Annessi. Backrunning Private Transactions Using Multi-Party Computatio, 2023. URL https://writings.flashbots.net/backrunning-private-txs-MPC. (cit. on pp. 1 and 3.)

Secret Network. Secret Network: A Privacy-Preserving Secret Contract & Decentralized Application Platform. URL https://scrt.network/graypaper. (cit. on p. 1.)

Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness Encryption and its Applications. 2013. URL https://eprint.iacr.org/2013/258.pdf. (cit. on p. 2.)

Dan Boneh, Amit Sahai, and Brent Waters. Functional Encryption: Definitions and Challenges. 2010. URL https://eprint.iacr.org/2010/543.pdf. (cit. on pp. 2 and 4.)

Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public Key Encryption with keyword Search. 2004. URL https://crypto.stanford.edu/~dabo/pubs/papers/encsearch.pdf. (cit. on p. 2.)

Alex Ozdemir and Dan Boneh. Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets. 2021. URL https://eprint.iacr.org/2021/1530.pdf. (cit. on p. 2.)

Stephan van Schaik, Alex Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Christina Garman, Daniel Genkin, Andrew Miller, Eyal Ronen, and Yuval Yarom. SoK: SGX.Fail: How stuff get eXposed. 2022. (cit. on p. 3.)

Daniel Moghimi. Downfall: Exploiting speculative data gathering. In *32th USENIX Security Symposium (USENIX Security 2023)*, 2023. (cit. on p. 3.)

Yulia Khalniyazova. MPC Solver Strategy Proof-of-Concept, 2023. URL https://github.com/anoma/mpc-solver-approx. (cit. on p. 3.)

Marcel Keller. MP-SPDZ. URL https://github.com/data61/MP-SPDZ/tree/master. (cit. on p. 3.)

Michael L. Dertouzos Ronald L. Rivest, Len Adleman. On data banks and privacy homomorphisms. 1978. URL https://luca-giuzzi.unibs.it/corsi/Support/papers-cryptography/RAD78.pdf. (cit. on p. 3.)