DSCS: Towards An Extensible Secure Decentralised Distributed Computation Protocol

Alexander Dalton
Imperial College London
a.dalton19@imperial.ac.uk

David Thomas
University of Southampton
d.b.thomas@southampton.ac.uk

Peter Cheung
Imperial College London
p.cheung@imperial.ac.uk

***Abstract*—This paper documents DSCS, a privacy preserving distributed computation protocol with native support for homomorphic encryption. The protocol is easily extensible with advanced cryptographic constructions due to a program encoding with a bijective map to arithmetic circuits. Arithmetic circuits are a common mechanism for abstract computation, and are widely used by advanced cryptographic constructions, such as homomorphic encryption and verifiable computation. This paper documents foundational work exploring the validity of an arithmetic circuit based approach to distributed computation. Ultimately, the direct use of arithmetic circuits as a computational substrate for distributed computation is shown to carry an untenable bandwidth cost, even under ideal network conditions. Importantly, this paper highlights a slew of low hanging fruit for further research in this domain, as well as guidance to avoid easy pitfalls.**

I. Introduction

The intention of this work is to define a secure-by-default decentralised and distributed computation protocol. Such a protocol would allow for mutually distrusting distributed devices, such as distributed sensors in a remote location, or Internet of Things (IoT) devices spread throughout a home, to collaborate on the execution of programs. Potentially, through collaboration with nearby devices the computational latency of individual task can be reduced, as devices leverage the spare computational capacity of their peers.

Fully Homomorphic Encryption (FHE) is used to provide data confidentiality in this work. Much like many other advanced cryptographic constructions, such as many Verifiable Computation (VC) constructions [1]–[3], FHE constructions operate over arithmetic circuits [4]–[6], mathematical objects that are abstractions of general computation. As such, any platform for computation that operates natively over arithmetic circuits can easily adopt new advanced cryptographic protocols. This work aims to explore the performance implications of such a protocol.

Alongside a protocol definition, a protocol implementation and programming framework supporting distributed computation paradigms has been created. This tool was used to glean experimental results and is publicly available on GitLab[1].

[1]https://gitlab.com/AlexDaltonUni/PhD/discus

A. Contributions

The contributions of this paper include:

- Proposing Distributed Secure Computation System (DSCS), an extensible decentralised distributed computation protocol over arithmetic circuits with native support for BFV, a Levelled Fully Homomorphic Encryption (LFHE) cryptosystem.
- Providing extensions to DSCS to harden it in escalating threat models:
  - A point-to-point token-based incentive scheme to encourage lazy network participants.
  - VC integration to assure program correctness.
  - Discussion around Fair Exchange (FE) considerations to maintain security against arbitrarily malicious adversaries.
- DiSCuS, a DSCS implementation and decentralised distributed computation framework.
- Quantitative performance measurements for DiSCuS.
- The identification of easy pitfalls of secure decentralised distributed computation protocol design centred around arithmetic circuits, and provide recommendations and guidance for further work.

To the best of our knowledge this constitutes the first exploration into the efficacy of a distributed computation protocol constructed from arithmetic circuits, the first distributed computation protocol designed from the onset with decentralisation in mind, and the first distributed computation protocol with native FHE support.

B. Threat Model

This work is presented in the context of a semi-honest adversarial threat model.

Definition 1 (Nosy Node Threat Model): Adversarial nodes will adhere to the protocol, but will attempt to passively learn confidential data. Nodes will enthusiastically engage with the tasks organised by other network nodes.

This threat model is reflective of a closed network of homogeneous devices, possibly distributed sensors in a remote location. Such a network consists of homogeneous devices wishing to exercise caution when operating over private data.

Protocol extensions are provided Appendix A and Appendix B in order to secure DSCS against progressively more powerful adversaries.

II. Background

A. Distributed Computation

Distributed computation networks are a class of usually-centralised distributed networks in which nodes collaborate on the execution of computational programs. There are a variety of protocols and frameworks built to support the operation of distributed computation networks, including: Charlotte [7], ParaWeb [8], ATLAS [9], and BOINC [10]. These each provide the component building blocks to construct a centralised distributed computational platform, including: isolated execution environments, program specification, and network primitives. These projects only support distributed computation in centralised networks, with vary degrees of minor protections against malicious participants.

The existing frameworks have been used to build operational distributed computation projects [10], [11]. Unfortunately, due to the constraints on the security model imposed by the components underpinning these distributed computation projects they are restricted to the domain of scientific computation, in which participants from around the world donate spare compute resource for the public good. One of the most well known examples of a distributed computation protocol is Folding@home [11], in which participants perform protein folding simulations to advance research in the field of organic chemistry. In the wake of the COVID19 health crisis [12] Folding@home saw an influx of volunteers and at it's peak was able to leverage 1.01 exaFLOPS of donated computational power [13].

B. LFHE

FHE [14] provides the ability to perform arbitrary computations over encrypted data. This work utilises a related class of cryptosystem, the LFHE, which adds the additional constraint that the number of operations must be known when the cryptosystem parameters are generated. Any computation that exceeds these bounds runs the risk of a decryption failure. LFHEs have the benefit of much lower computational costs than less restrictive FHE. The BFV LFHE cryptosystem [4] is chosen as a specific LFHE in the DSCS implementation due to it's popularity and compatibility with Number Theoretic Transform (NTT) acceleration techniques.

C. NTT FHE Acceleration

Residue Number System (RNS) acceleration techniques are a popular mechanism used to transform a sequence of arithmetic operations modulo a large integer, into an independent sequence of modular arithmetic operations over smaller fixed precision integers [15]. The NTT extends the RNS approach from integers to large polynomials. Many LFHE cryptosystems use large polynomials to encode ciphertexts [16], and therefore NTT can be used to accelerate LFHE evaluation. Often in related work the application of NTT encodings to polynomials are also referred to interchangeably as the Double Chinese Remainder Transform, and simply as RNS.

This transformation maps a single element from a ring of polynomials with large integer coefficients taken modulo a large composite number $N = \prod_{p \in P_N} p$, into a set of $|P_N|$ polynomials each with coefficients taken modulo a distinct element of P_N. These polynomials can then be operated on independently, and recombined to produce a result equivalent to processing the original polynomials in their original big-integer encoding. The interested reader is directed to [16] to explore this technique further.

The LFHE used in this work, BFV, works well with RNS-style acceleration for it's homomorphic addition operations. Unfortunately multiplication is more complex, and does not map directly onto operations on the underlying polynomial ring, as such it requires additional machinery, as described in [17].

D. Security Properties

First, there is the problem of operating over private data. Consider computational tasks conducted by IoT devices, such as home security systems or smart locks, which frequently operate over sensitive data. Any competent secure computation protocol should support the confidentiality of private data.

Definition 2 (Configurable Confidentiality): User-defined programs support the ability to flag operands as private, and other network participants are unable to view this data, or any of it's derivatives.

A common sense protocol requirement is that any computational tasks accelerated by the network should produce the correct result.

Definition 3 (Computation Soundness): The results of computations must be correct.

E. Performance

In order to set a performance target we consider the idealised behaviour of an NTT accelerated LFHE-based computation protocol. Under ideal conditions we would expect the latency of an accelerated program to conform to the following equation:

$$t_{\mathsf{DSCS}} = \frac{ec_e + c_p}{n} + c_i + o_n n + \mathsf{overhead}(c_e + c_p + c_i) \quad (1)$$

where e is the expected expansion in computational effort for BFV evaluation; c_e, c_p, and c_i are the abstract computational costs for the homomorphically encrypted, unencrypted parallelisable, and unencrypted iterative segments of a target program respectively; o_n is a per-node overhead; n is the number of participating nodes; and $\mathsf{overhead}$ is a weighted linear combination of all the instructions within a target program, where weights are per-instruction overheads. Given $\mathsf{overhead}$ is sufficiently small performance is expected to approximate Figure 1, where t_{local} is the comparison to local-only execution given by:

$$t_{\mathsf{local}} = ec_e + c_p + c_i. \quad (2)$$

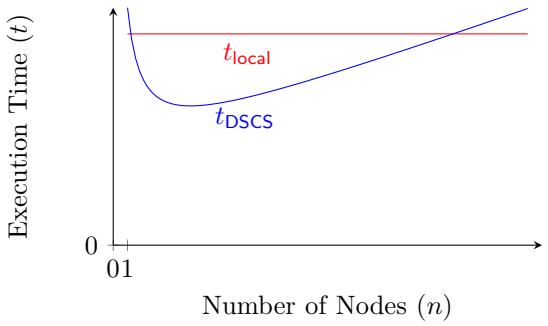

Figure 1. The theoretical latency of a computational task delegated to the DSCS network.

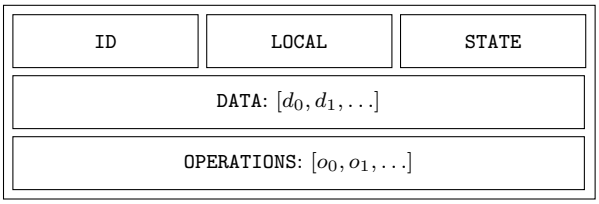

Figure 2. DSCS Work Fragment.

Note that the FHE encrypted portion, ec_e, is still FHE encrypted, as we assume that the devices still wish to utilise the increased resistance to physical tampering provided by FHE.

More concretely, our performance target is that for some number of nodes the latency of a computational task delegated to the network is less than the latency of the same computational task executed locally.

Definition 4 (Performance Target): For some n the latency of executing a computational task is lower when delegated to the network than with local-only execution:

$$t_{\text{DSCS}} < t_{\text{local}}. \tag{3}$$

III. DSCS Protocol

The DSCS protocol supports general computation between nodes in a decentralised and distributed network with the intention of reducing the latency of individual computations through inter-node collaboration. This feature is intended to drive the adoption of FHE in contexts where the cost would otherwise be prohibitive.

A. Work Fragments

Programs on the DSCS network are expressed as a set of *Work Fragments*, each of which describes a self-contained portion of the program. The structure of a *Work Fragment* is shown diagrammatically in Figure 2. *Work Fragments* can be assembled in advance of execution or assembled dynamically to better suit a variety of network conditions.

Within a *Work Fragments* the `ID` provides an identifier generated by the originating node, this is used to correlate incoming results with the original program. A flag `LOCAL` identifies this *Work Fragment* as containing unencrypted private information if true. The `STATE` can be one of four values: `WAITING` when there are unresolved dependencies, `READY` when all external dependencies have been resolved, `IN PROGRESS` when the *Work Fragment* is currently being evaluated, and `DONE` when evaluation terminates. `DATA` is an ordered set of write-once data which can be one of four types: an unsigned integer, BFV ciphertext, a fragment of an NTT-encoded BFV ciphertext, or a special `Wait` type. `Wait` indicates the operand is the output of a yet-to-be-executed operation. `OPERATIONS` is an ordered set of arithmetic instructions to be executed in-turn. Arithmetic operations supported include: `ADD`, `SUB`, `MUL`, and `DIV`. By design *Work Fragments* have a fixed and easily pre-calculable execution cost. Some of the elements of `DATA` are designated as the output of the *Work Fragment*, upon completing *Work Fragment* evaluation data in these locations is returned to the originating node.

The operations can accept operands of mixed-types, this functionality is achieved by embedding public keys in the BFV ciphertexts, allowing plaintext values to be encrypted ad-hoc. For a given instruction, if at least one operand is an RNS-encoded BFV ciphertext fragment the output of an operation will be an RNS-encoded BFV ciphertext fragment, if not and at least one operand is a BFV ciphertext the output of an operation will be a BFV ciphertext, otherwise the output of the operation is an unsigned integer.

Work Fragments are evaluated by executing each operation listed in `OPERATIONS` in turn, iteratively mutating `DATA` until each instruction has been executed. The vector of all data marked as a result is collated and returned to the originating node.

B. Protocol Description

Each node has a public identity which corresponds to their public key for a pre-agreed signature scheme. All messages are cryptographically signed via this signature scheme, allowing the originator of all messages to be verified. Nodes are organised in an ad-hoc topology with translation between public identifiers and IP addresses provided by an Multicast DNS (mDNS) protocol which runs in parallel with the DSCS protocol.

Nodes construct short lived centralised subnetworks to collaborate on a single task. Each node organises a subnetwork of collaborators for, at most, one in-progress task under their ownership. The set of nodes collaborating on a given task are referred to as a *Work Group*. Regardless of the existence of any in-progress tasks, DSCS requires each node to provide four public functions:

- **Advertise**: $\top$ or $\bot \leftarrow \mathsf{Adv}()$; returns $\top$ if this node is organising an in-progress task, and $\bot$ otherwise.
- **Registration**: $\top$ or $\bot \leftarrow \mathsf{Reg}(i)$; registering node with identity i to the current *Work Group*. Returns $\top$ if successful, and $\bot$ if unsuccessful. i must be the identity of the node making the request.

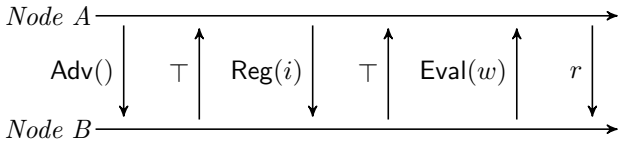

Figure 3. Node A inquires after any work Node B is advertising, Node A then registers for Node B's *Work Group*. Node B then sends Node A *Work Fragment* w for evaluation, who returns result r.

- **Deregistration**: $\top \leftarrow \mathsf{Dereg}(i)$; unregister a node with identity i from the current *Work Group*. i must be the identity of the node making the request.
- **Evaluate**: r or $\perp \leftarrow \mathsf{Eval}(w)$; if *Work Fragment* w originates from the organiser of a registered *Work Group*, evaluate w and return result r, otherwise return $\perp$.

Independent nodes are free to interact with these public-facing functions however they wish. The threat model assumes that nodes will enthusiastically engage with the tasks proposed by their peers, actively seeking out *Work Groups* and driving network computation throughput. An end-to-end example including joining a *Work Group* and evaluating *Work Fragments* is provided in Figure 3.

C. Protocol Evaluation

Clearly this protocol is both distributed and decentralised, but the required security properties warrant some further discussion.

Configurable Confidentiality is achieved via two measures. First, confidential data is suitably flagged, and as such any *Work Fragments* containing confidential data in plaintext have the `LOCAL` flag set to true. *Work Fragments* with the `LOCAL` flag set to true should only ever be evaluated locally, and as such never leave the originating node. The results of computations over confidential data are themselves confidential, and the persistence of confidentiality is achieved through poisoning the data with the confidentiality flag. The second measure deployed to ensure data confidentiality is the BFV LFHE. The originating node is able to set the `LOCAL` flag to false by encrypting any confidential data via BFV, parallelising the homomorphic operations via the NTT to produce many independent *Work Fragments* that are able to drive the processing of confidential data on remote nodes.

Violating the *Computation Soundness* property is not possible in this threat model. The protocol extensions provided in Appendix B ensure this property is supported by cryptographic machinery, rather than adversarial assumptions.

IV. DiSCuS

DiSCuS is an implementation of the DSCS protocol and distributed computation framework, intended to demonstrate the efficacy of a secure decentralised distributed computation platform. It is written in Rust and implements DSCS as it is defined the previous section. The insights gleaned from this work are detailed towards the end of this paper and should act as useful guidance for subsequent research.

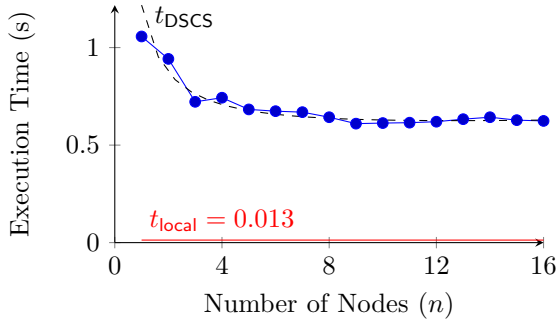

Figure 4. Latency of 1600 equivalent-plaintext-addition-operations via independent homomorphic additions on the DiSCuS network. The line t_{DSCS} is fit against the data via polynomial regression with $R^2 = 0.92$.

A. Performance Evaluation

In order to provide a sanity check of the optimistic best-case performance for the system a simple benchmark was constructed featuring 1,600 independent homomorphic ciphertext additions. Due to SIMD properties of BFV, with a ciphertext ring size set arbitrarily at 16, 1,600 independent addition operations translates into 100 ciphertext operations. The benchmark was run on a single desktop computer with a multicore CPU, with DSCS networks consisting of between 1 and 16 nodes, where results were averaged over 30 independent runs. A single desktop computer was used to provide best-case network latency. These results were compared with an equivalent program executed locally with the same DSCS instruction interpreter, bypassing the DSCS network entirely. In both cases the benchmark measures the time between the first instruction beginning evaluation (i.e. leaving the host node) and receiving the ultimate result. This benchmark should highlight the best-case performance of the system under ideal network conditions.

The experimental results in Figure 4 paint disappointing picture as compared with the target performance stated in Figure 1. Using polynomial regression to extract constants for Equation 2 we get $c = 0.697$, $o_n = 0.004$, and $\mathsf{overhead}(c) = 0.00074c$. The results of this best-case benchmark highlight some fundamental flaws with the design of DSCS.

There are three primary factors contributing to poor performance:

1) The time required to transmit a *Work Fragment* is a significant factor in the execution latency. As they exist *Work Fragment* do not offer a compressed enough program encoding.
2) The current *Work Fragment* format requires that during runtime the node continuously checks for unblocked dependencies.

3) The data stored in the runtime *Work Fragment* data pool is write-once, requiring that a large volume of `Wait` operands are transmitted to satisfy all intermediate results.

V. Future Work

This body of work provides us with clear guidance for any future work attempting to design a tenable decentralised distributed secure computation protocol.

- A more compressed program representation must be designed, transmission time can not be an overwhelming element of execution latency. This could be achieved via:
  - The introduction of a `LOOP` instruction to repeat identical subtrees within the arithmetic circuit.
  - Replacing *Work Fragments* with a new more compressed arithmetic circuit encoding.
  - Replacing arithmetic circuits entirely for a more expressive program encoding. Control flow and boolean operations will allow for a much more expressive program encoding, although diverging from arithmetic circuits will cause difficulties for the incorporation of any advanced cryptographic constructions.
- In order to relieve constraints on runtime dependency management, instructions in program encodings should include information about any instructions they are blocking. Alternatively a more robust execution runtime with operand caching and intelligent dependency management could alleviate some of these concerns.
- Data pools accompanying program encodings should no longer be write-once to allow reusing data locations and therefore enabling more compact program encodings. The bijective mapping between arithmetic circuits and *Work Fragments* will be enforced by a compiler capable of generating *Work Fragments*.

The primary concern of future implementation work must be program compactness. The time taken to disseminate atomic program units to collaborating nodes should be a negligible part of the execution latency.

VI. Conclusion

There is a fair amount of work to be done in improving DSCS, or creating a DSCS-like protocol, to produce a compelling computational substrate for mutually distrusting devices to securely collaborate on tasks. In this current incarnation there is no reason for a device to invest the effort into organising a group of peers over performing a computation locally, in isolation.

There are a slew of low-hanging-fruit architectural and implementation improvements that can be made to DSCS, which in conjunction could conceivably result in massively reduced operational overheads. The primary objective of further research must be to producing much more compact program encodings. If this can be achieved it is certainly within the realm of possibility that a system like DSCS could have performance such that it is able to provide utility over local-only execution.

References

[1] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, May 2013, best Paper Award. [Online]. Available: https://www.microsoft.com/en-us/research/publication/pinocchio-nearly-practical-verifiable-computation/

[2] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," Cryptology ePrint Archive, Paper 2017/1066, 2017, https://eprint.iacr.org/2017/1066. [Online]. Available: https://eprint.iacr.org/2017/1066

[3] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in *Advances in Cryptology – CRYPTO 2019*, A. Boldyreva and D. Micciancio, Eds. Cham: Springer International Publishing, 2019, pp. 733–764.

[4] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[5] A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," Cryptology ePrint Archive, Report 2013/094, 2013, https://eprint.iacr.org/2013/094.

[6] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017, pp. 409–437.

[7] A. Baratloo, M. Karaul, Z. Kedem, and P. Wijckoff, "Charlotte: Metacomputing on the web," *Future Generation Computer Systems*, vol. 15, no. 5, pp. 559–570, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X99000096

[8] T. Brecht, H. S. Sandhu, M. Shan, and J. Talbot, "ParaWeb: towards world-wide supercomputing," in *EW 7*, 1996.

[9] J. E. Baldeschwieler, R. D. Blumofe, and E. A. Brewer, "ATLAS: An infrastructure for global computing," in *EW 7*, 1996.

[10] D. Anderson, "BOINC: A system for public-resource computing and storage," in *Fifth IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 4–10.

[11] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," in *2009 IEEE International Symposium on Parallel & Distributed Processing*, 2009, pp. 1–8.

[12] M. Ciotti, M. Ciccozzi, A. Terrinoni, W.-C. Jiang, C.-B. Wang, and S. Bernardini, "The COVID-19 pandemic," *Critical reviews in clinical laboratory sciences*, vol. 57, no. 6, pp. 365–388, 2020.

[13] M. I. Zimmerman, J. R. Porter, M. D. Ward, S. Singh, N. Vithani, A. Meller, U. L. Mallimadugula, C. E. Kuhn, J. H. Borowsky, R. P. Wiewiora *et al.*, "SARS-CoV-2 simulations go exascale to predict dramatic spike opening and cryptic pockets across the proteome," *Nature Chemistry*, vol. 13, no. 7, pp. 651–659, 2021.

[14] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.

[15] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, *Residue number system arithmetic: modern applications in digital signal processing*. IEEE press, 1986.

[16] V. Lyubashevsky, C. Peikert, and O. Regev, "A toolkit for Ring-LWE cryptography," in *Advances in Cryptology – EUROCRYPT 2013*, T. Johansson and P. Q. Nguyen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 35–54.

[17] S. Halevi, Y. Polyakov, and V. Shoup, "An improved RNS variant of the BFV homomorphic encryption scheme," in *Topics in Cryptology – CT-RSA 2019*, M. Matsui, Ed. Cham: Springer International Publishing, 2019, pp. 83–105.

[18] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in Cryptology – CRYPTO 2010*, T. Rabin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 465–482.
[19] A. Küpçü and A. Lysyanskaya, "Usable optimistic fair exchange," *Computer Networks*, vol. 56, no. 1, pp. 50–63, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912861100301X
[20] H. Pagnia and F. C. G. Darmstadt, "On the impossibility of fair exchange without a trusted third party," 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:11671049
[21] A. Dalton, D. Thomas, and P. Cheung, "Two party fair exchange," Cryptology ePrint Archive, Paper 2023/585, 2023, https://eprint.iacr.org/2023/585. [Online]. Available: https://eprint.iacr.org/2023/585

Appendix

A. Protocol Extension for Lazy Nodes

Conceivably, adversaries may try to exploit the protocol in order to leverage more computation from the network than they are willing to return in kind. This would be achieved by engaging lazily in the work of their peers. Adversarial nodes such as these are self-motivated and would act to maximise their utility.

To defend against lazy adversarial behaviours a node-to-node credit system is introduced. Each node maintains a signed integer, reflecting a credit total, for each node they interact with. A positive value representing being in credit, and therefore can easily leverage the other node for compute, a negative value representing debt which could be redeemed at any time. Completed work is rewarded with credit, which is non-transferable and can only be redeemed at the originating node for the evaluation of *Work Fragments* for an equitable amount of work. The credit awarded for evaluating a *Work Fragment* is given by the weighted total of all `OPERATIONS` contained within, with weights predefined network-wide. A node's credit can be redeemed for work at the credit-issuing node.

B. Protocol Extension for Dishonest Nodes

Arbitrarily malicious nodes suggest four primary categories of adversarial activity:

1) **Falsification of results**, where a remote node returns incorrect results for the evaluation of a *Work Fragment*.
2) **Falsification of credit**, where a node attempts to redeem unearned credit, or a remote node denies an attempt to redeem honestly earned credit.
3) **Unfair exchange of credit**, where a node will try to gain credit without releasing the results of a *Work Fragment*, or a node attempts to gain the results of a *Work Fragment* without releasing credit.
4) **Clean identity generation**, where nodes attempt replace identities associated with malicious activity with a new identity.

Falsification of results is easily combated with the addition of a VC scheme to the system [18]. Before being sent to a remote node for evaluation, *Work Fragments* are preprocessed via VC.ProbGen, and then upon receiving the results of the evaluated *Work Fragment* the results are extracted via VC.Vf.

Combating **falsification of credit** requires extending the credit system such that any third party can ascertain the truthfulness of any allegations of dishonesty. To this end whenever the credit value changes, the node which is losing credit (either because they are awarding credit, or because they are spending it on an outsourced evaluation) cryptographically signs a concatenation of the new credit total with a strictly increasing integer ID. This is then sent to the other node. If a dispute occurs at a later date, an honest node will be able to present a signed credit total with the largest ID to corroborate their claims.

In order to remove the possibility of the **unfair exchange of credit** an FE scheme must be used whenever credit is exchanged for the results of an outsourced computation [19]. Unfortunately, most FE constructions require a trusted third party, and so resist decentralisation [20]. There exists some notion of two party FE, but it remains well beyond the realm of practicality [21]. Regardless, even if two party FE was readily available, there is no way to naively compose FE and VC, as using FE is equivalent to ensuring each transaction is atomic, and as such the credit-issuing node is unable to run VC.Vf before issuing the credit.

Clean identity generation occurs when a node wishes to discard an identity associated with malicious behaviour and adopt a new identity. There is a subtle pressure in DSCS that decentivises frivolously discarding identities provided by the credit system. Nodes will be more likely to trust long lived identities to persist long enough for any issued credit to be redeemed. As such new nodes on the network will find themselves having to commit computational effort to unchoking themselves by earning credit from their peers. Unfortunately, this is a heuristic defence at best.