

Towards Secure Distributed Computation

Alexander Dalton
Imperial College London

Supervised by David Thomas and Peter Cheung

Submitted in part fulfilment of the requirements for the degree of Doctor of Philosophy in Electrical and
Electronic Engineering of the Diploma of Imperial College, December 2023

Abstract

Distributed computation is a computational paradigm in which a number of discrete computers are connected via a network and collaborate on the execution of programs. These systems exist in a variety of contexts, such as: volunteer scientific computation, blockchain, and file sharing swarms. Many existing systems assume altruistic behaviour from their participants. As such, a number of open questions exist: Can network protocols be orchestrated in such a way as to encourage altruistic behaviour from nodes that would otherwise be unmotivated to collaborate? How could a decentralised computation network be hardened against adversarial actors interested in disrupting computations? Are there any incentives that need to be put into place to encourage altruistic behaviour in otherwise self-motivated network participants?

The main identified vulnerabilities in a distributed computation protocol identified in this work are:

1. Correctness of delegated program results.
2. Discouraging self-motivated behaviours.
3. Confidentiality of private data.

This work provides a class of decentralised and distributed computation protocol which supports all three: a Trusted Computation Mesh (TCM), with an exemplar TCM developed as part of this work named Distributed Secure Computation System (DSCS).

This thesis aims to provide a specification for a secure-by-default decentralised distributed computation scheme, in which non-homogeneous untrusted nodes can organically distribute work. There are a variety of contributions; new cryptographic constructions and protocol formulations, as well as a reference protocol implementation. Most of the contributions are purely theoretical in nature, with some claims supported by implementation work later on. A number of supporting technologies are created or, where solutions already exist, improved upon. These technologies are later incorporated into the decentralised distributed computation network protocol DSCS and the corresponding implementation DiSCuS.

Dedications

First and foremost, thank you to my fantastic supervisors, David Thomas and Peter Cheung, without whom I would never have been able to produce this work. Early in my PhD both impressed on me the importance of following your nose to whatever fruitful research topic was of most interest and had the greatest potential to yield interesting results. Without this advice, this thesis would have been a much less interesting read. David deserves special thanks for to the tireless diligence with which he faced an endless pile of drafts of both papers and thesis chapters. His notes are always actionable and precise, and I never left our meetings feeling anything less than enthusiastic to continue my work. To both David and Peter I can only apologise for stretching my original research proposal to its absolute limits and then discarding it entirely.

My deepest thanks to all the members of the Circuits and Systems research group, the academic staff, administrative staff, and my fellow PhD students. You're a great bunch and did all that you could to squash the impostor syndrome felt by every fledgling PhD student. I can only apologise for the deeply theoretical nature of my work. You all have an standing invitation to visit with me, to dust off some board games and to sample my cooking.

A large fraction of my time as a PhD student was spent living with four of my closest friends: Crosby, Joey, Joe, and Barny. They are owed my deepest thanks and nothing less than a sincere apology for the unsociable hours I kept and the times I was caught pacing around our home at odd times of the night as I worked through some obscure problem. In particular, I'm forever indebted to Barny, one of my longest suffering and closest friends, to whom I have delivered more soliloquies on my work than anyone else. Such has been his service that his knowledge of my research probably rivals that of my supervisors'. A huge thank you is also owed to Sam and Alice, the too infrequent trips to their home and subsequent excursions into the countryside did a lot to balance my humours during this process.

I owe a huge debt of gratitude to my partner Beth, who has supported me in ways both minor and massive over the last 8 years. Her patience and rock-solid kindness have made the hardest days nothing more than a cross on the calendar. If I can support you though your PhD half as well as you have supported me, you'll breeze through. Between us we've had more than a couple of dead-end jobs and advanced degrees, and hopefully we don't have too many more of either. And to Beth's family: Ian, Sarah, and Ben, who have shouldered more than their fair share of my burdens, making sure I was fed and watered during my time as an undergraduate at the University of Bristol, thank you.

Last but not least, I could not have produced this work were it not for my parents' unbroken 28-year-running streak of faith in me. Their unrelenting belief in my capabilities has been so consistent that through my youth I often mistook it for an irritating unwillingness to accept the facts of reality. A straight line can be drawn from a childhood spent with my Dad doing before-bed arithmetic to the ability to do the research presented in this thesis. Mum, you always said I would write a book, and for now this will have to do. To my younger brothers: Tate, Nathan, and Joseph - I hope you're as proud of me as I am of each of you.

Statement of Originality

The work contained within this thesis is entirely my own, and all other work is appropriately referenced.

Copyright Declarations

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Contents

List of Tables	xv
List of Figures	xvii
List of Definitions	xxiii
1 Introduction	1
1.1 Motivations	2
1.1.1 Applications	3
1.2 Motivating Adversarial Behaviours	4
1.2.1 Corrupting Computation	4
1.2.2 Avoiding Work	5
1.2.3 Unfair Exchange	5
1.2.4 Viewing Private Data	5
1.3 Challenges	5
1.4 Contributions	6
1.5 Publications	7
1.6 Thesis Structure	7
2 Background	9
2.1 Notation	9
2.2 Distributed Networks	9
2.2.1 Distributed Computation	11
2.3 Residue Number Systems	12
2.4 Cryptography	13
2.4.1 Encryption Schemes	14
2.4.2 Proofs by Reduction	16
2.5 Conclusion	17
3 Proofless Verifiable Computation from Integer Factoring	19

3.1	Introduction	20
3.1.1	Contributions	21
3.1.2	Threat Model	22
3.1.3	Assumptions	22
3.2	Background	22
3.2.1	Notation	23
3.2.2	Verifiable Computation	23
3.2.3	Arithmetic Circuits	25
3.2.4	Integer Factoring Assumption	26
3.3	Related Work	27
3.3.1	Auditing	28
3.3.2	Trusted Platform Modules	28
3.3.3	Number Theoretic Solutions	28
3.3.4	Identified Problems	30
3.4	Intermediate Verifiable Computation Scheme	30
3.4.1	Outline	32
3.4.2	Definition	32
3.4.3	Correctness and Outsourcability	32
3.4.4	Soundness	33
3.5	Verifiable Computation Scheme from the Hardness of Integer Factoring	35
3.5.1	Outline	35
3.5.2	Protocol Definition	36
3.6	Evaluation	36
3.6.1	Completeness	36
3.6.2	Soundness	38
3.6.3	Outsourcability	40
3.6.4	Performance Implications of the Factoring Problem	41
3.6.5	Proof Size	42
3.6.6	Adapting General Problems	42
3.6.7	$\mathbb{Z}_N$ Too Large	42
3.6.8	$\mathbb{Z}_N$ Too Small	43
3.7	Conclusion	43
4	Two Party Fair Exchange	47
4.1	Introduction	48
4.1.1	Contributions	48
4.1.2	Threat Model	48
4.1.3	Assumptions	49
4.1.4	Additional Assumption for Initial Protocols	51
4.2	Background	51
4.2.1	Notation	51
4.2.2	Fair Exchange	51
4.2.3	Symmetric IND-CPA Secure Authenticated Encryption Schemes	53
4.2.4	Oblivious Transfer	54

4.2.5	Non-Interactive Zero Knowledge Proofs (ZKPs)	54
4.2.6	Commitment Schemes	54
4.3	Related Work	55
4.4	Extended Definitions for Fair Exchange	55
4.5	Candidate Approaches	56
4.5.1	Onion Encryption	56
4.5.2	Leaking Keys Through Position of Secrets	57
4.5.3	Shared State	57
4.6	Sharing a Single Random Bit	58
4.6.1	Description	58
4.6.2	Evaluation	59
4.7	Sharing a Single Chosen Bit	59
4.7.1	Description	59
4.7.2	Attack Against Computational Strong Fairness	62
4.7.3	Evaluation	62
4.8	Exchanging Secrets of Arbitrary Length	63
4.8.1	Description	64
4.8.2	Proofs	65
4.8.3	Evaluation	72
4.9	Two Party Fair Exchange with Malicious Adversaries	73
4.9.1	Additional Assumptions	73
4.9.2	Protocol Extensions	73
4.9.3	Properties	74
4.9.4	Evaluation	78
4.10	Asymmetry Between X and Y	78
4.11	Applications to Consensus	79
4.11.1	The Consensus Problem	79
4.11.2	Existing Consensus Schemes	80
4.11.3	Constructing Consensus Protocols from Two Party Fair Exchange	80
4.11.4	Evaluation	80
4.12	Further Work	81
4.13	Conclusion	82
5	DSCS: A Trusted Computation Mesh	83
5.1	Introduction	83
5.1.1	Contributions	84
5.1.2	Assumptions	84
5.1.3	Performance Assumptions	85
5.2	Related Work	85
5.3	Background	86
5.3.1	Verifiable Computation	86
5.3.2	Two Party Fair Exchange	87
5.3.3	Homomorphic Encryption	88
5.3.4	Signature Schemes	89

5.3.5	Threat Model	91
5.4	Secure Multi-User Systems	92
5.5	Trusted Computation Mesh	94
5.6	DSCS in the Nosy Node Threat Model	96
5.6.1	Overview	98
5.6.2	Preliminary Definitions	100
5.6.3	Protocol Definition	104
5.6.4	Properties	108
5.7	DSCS in the Lazy Node Threat Model	109
5.7.1	Extended Protocol Definitions	111
5.7.2	Properties	112
5.8	DSCS in the Untrusted Node Threat Model	116
5.8.1	Defences Against Falsification of Results	117
5.8.2	Defences Against Credit Falsification	118
5.8.3	Defences Against The Unfair Exchange of Credit	121
5.8.4	Defences Against Identity Regeneration	122
5.8.5	DSCS State Machine in an Untrusted Network	123
5.8.6	Properties	124
5.9	Evaluation	128
5.10	Further Work	132
5.11	Conclusion	134
6	DiSCuS: A Partial DSCS Implementation	135
6.1	Introduction	135
6.1.1	Contributions	137
6.2	Background	137
6.2.1	Notation	138
6.2.2	DSCS	138
6.2.3	BFV	139
6.2.4	Double CRT Ciphertext Representation	140
6.3	DiSCuS	143
6.3.1	Mapping Conventional Arithmetic onto Homomorphic Evaluation	144
6.3.2	Algorithms	144
6.3.3	Programs	145
6.3.4	Interpreter	146
6.3.5	Implementation	148
6.4	Evaluation	152
6.4.1	DiSCuS Performance	153
6.5	Related Work	159
6.6	Further Work	160
6.6.1	Runtime	160
6.6.2	Compiler	160
6.6.3	DSCS Protocol	160
6.6.4	Speculative Research	160

6.7	Conclusion	161
7	Conclusion	163
7.1	Objectives	163
7.1.1	Proofless Verifiable Computation from Integer Factoring	163
7.1.2	Two Party Fair Exchange	164
7.1.3	DSCS: A Trusted Computation Mesh	165
7.1.4	DiSCuS: A Partial DSCS Implementation	166
7.2	Towards Secure Distributed Computation	167
7.3	Challenges	168
7.4	Future Work	168
7.5	Conclusion	169
	Bibliography	171
A	DiSCuS Experimental Results	181
A.1	Independent Plaintext Addition	181
A.2	Independent Homomorphic Encryption (HE) Addition	183
A.3	Independent HE Addition in DCRT Form	183
A.4	Dependent HE Addition	183
A.5	Dependent HE Addition in DCRT Form	183
B	DiSCuS Polynomial Regression	185

List of Tables

3.1	Computational complexity of a selection of state of the art Verifiable Computation (VC) schemes, compared to unverified remote computation, and naive duplication. x is the input to arithmetic circuit C with depth d and result r . Schemes over boolean circuits use B instead of C	31
3.2	Related work Table 3.1 extended to include the new VC scheme presented in this work. . . .	44
5.1	Comparison of some cryptosystems with homomorphic properties	90
5.2	DSCS heuristic credit weightings. These heuristics only loosely reflect reality, and the normalised computational burden of each operation is dependent on external factors such as cryptosystem choice and parameters, and availability of hardware coprocessors.	111
5.3	Secure Multi User System (SMUS) feature comparison table	129
6.1	Benchmark figures for DiSCuS applications.	157
6.2	Benchmark figures for DiSCuS applications normalised against Local (Interpreted) results. . .	157
6.3	Speed-up of 16 node <i>Work Group</i> over 1 node <i>Work Group</i>	158
6.4	Constants extracted from experimental results by polynomial regression.	158
A.1	Independent plaintext addition experiment results.	182

List of Figures

1.1	Distributed computation protocol motivating example: sensors in a field, IoT devices in a home, and a data centre deployment.	3
1.2	Dependencies between the background and content chapters in this thesis, with $A \rightarrow B$ indicating Chapter B builds on the content of Chapter A	8
2.1	Indistinguishability under Chosen Plaintext Attack (IND-CPA) security game. (1) A key, k is generated, as well as random bit b . (2) The adversary generates two messages. (3) m_b is encrypted. The adversary $\mathcal{A}$ wins if they can correctly identify b given the encryption of m_b . The adversary is also given access to an encryption oracle OEnc_k	15
2.2	Hypothetical adversary $\mathcal{A}$, capable of winning the IND-CPA security game as detailed in Figure 2.1.	16
2.3	Adversary $\mathcal{B}$; capable of winning the Indistinguishability under Chosen Ciphertext Attack (IND-CCA) security game given the existence of adversary $\mathcal{A}$	16
3.1	VC protocol overview. A public and private key pair is generated, a target function f and input x are encoded as problem σ , and this is sent to the Prover. The Prover then evaluates the problem, generating output r , which is sent to, and verified by, the Verifier.	24
3.2	Verify security game for VC schemes. (1) Generate key pair, (2) send f and pk to adversary, which returns an encoded problem σ and forgery r' , (3) the adversary wins if r' is not the correct result of $\text{Compute}(\sigma)$ but is accepted by the Verifier regardless.	24
3.3	KGen, ProbGen, Compute, and Vf for a trivial but non- <i>Outsourceable</i> VC scheme.	25
3.4	An arithmetic circuit representation of $r = c(a + b)$	26
3.5	Factoring security game for the integer factoring problem. (1) Generate suitably hard factorisation problem, (2) send N to the adversary and receive result a , (3) the adversary wins if a is a non-trivial factor of N	27
3.6	The KGen, ProbGen, Compute, and Vf function definitions for the initial VC scheme.	33
3.7	Adversary $\mathcal{A}$; capable of producing forged results r' in this VC scheme to VerifyGame , as defined in Figure 3.2, with advantage $\text{negl}(\lambda) < \text{Adv}_{\mathcal{A}}^{\text{Vf}}(\lambda) < 1$	34

3.8	Adversary $\mathcal{B}$, capable of calculating two non-trivial factors to any composite integer given the existence of Adversary $\mathcal{A}$	35
3.9	The KGen, ProbGen, Compute, and Vf function definitions for the VC scheme, where KGen has access to an oracle capable of formulating the integer factoring problem for a given security parameter, PrimeFactorProblem(λ).	37
3.10	VC from the hardness of integer factoring protocol overview. This can be derived by applying the function definitions in Figure 3.9 to the exchange detailed in Figure 4.6 with some of the Vf functionality precalculated by the Verifier as they await r from the Prover.	37
4.1	The NIND-CPA security game, in which an adversary must correctly identify the element in set S which corresponds to an encryption of m_1 , where all the other elements in set S are an encryption m_0 . There is an implicit state maintained between the two calls to adversary $\mathcal{A}$	50
4.2	Trivial Fair Exchange (FE) protocol with a trusted third party arbitrator, A	52
4.3	Strong Fairness security game for an FE protocol. δ is a potentially partial protocol transcript randomly sampled from the space of possible transcripts Δ , and $\mathcal{A}_P$ is an adversary acting as party P . If the adversary is able to extract the data of the other participant, d_Q , they win. P and Q are used to allow the adversary to distinguish between the specific rolls X and Y have in the given FE schemes.	56
4.4	Function definitions for the chosen single-bit two party FE scheme. For notation simplicity it is assumed that variables are name-spaced to their own function.	60
4.5	Two party FE protocol for chosen single-bit secrets.	60
4.6	Overview of a single round of the two party FE protocol supporting the exchange of arbitrarily sized secrets. The round transcript δ is specified at each stage of communication for use in later proofs.	66
4.7	For an FE protocol between arbitrarily labelled parties P and Q , with a set of possible protocol transcripts Δ , transcript $\delta \in \Delta$, and adversary acting as party P given by $\mathcal{A}_P$. If the adversary is able to extract the data d_Q from δ , they win.	67
4.8	The PFairnessGame from Figure 4.7 adapted to model repeat rounds of the protocol. The adversary acting as X has access to an oracle, OTranscript(d_Y), which provides random FE protocol transcripts of length 3 (i.e. $\delta = \{c_Y, c_X, c_z\}$). d_Y is initially sampled from the set of all possible input data D , and is then used to instantiate the oracle.	69
4.9	Hypothetical adversary $\mathcal{A}_X$ for the bounded RRXFairnessGame.	70
4.10	An adversary $\mathcal{B}$ capable of winning the NINDCPAGame given an adversary $\mathcal{A}$ capable of winning the n -bounded RepeatXFairnessGame. Whenever $\mathcal{A}$ requests a sample transcript the c_Y component is set to an encryption of the index i . When $\mathcal{A}$ is able to correlate the key used to encrypt the data they extract and return the index.	70
4.11	Overview of a single round of the two party FE protocol extended to be sound against malicious adversaries. Helper subroutines ConstructVX and ConstructVCY are defined in Figure 4.12. . .	75
4.12	Helper subroutines for constructing $\mathbf{v}_X, \mathbf{v}_Y$, and $\mathbf{c}_Y$	76

4.13	the PExtFairnessGame security game. First random challenge data is sampled, d , then the adversary $\mathcal{A}$ is allowed to generate a partial transcript δ which only contains the messages that would be sent by one of the two participants. This transcript is then simulated by Simulate , which fills in δ with the corresponding messages of the other party until that party would terminate early given the protocol definition in Figure 4.11. Simulate returns $(\delta', d_{\mathcal{A}})$ where δ' is the transcript filled until an error was reached and $d_{\mathcal{A}}$ is the data “given up” by $\mathcal{A}$ during the simulation of δ	76
4.14	Consensus protocol from two party FE as defined in [93] in which two nodes can reach consensus over a 1-bit value, $v \in \{0, 1\}$. The construction requires Desc , a function capable of producing a description of expected data, and FE which is a two party FE invocation that takes the sent secret, d_X , as the first argument and a description of the received secret, Desc (d_Y), as the second argument. With honest participants this consensus construction returns 1 if both parties input 1, and 0 otherwise. When interacting with a dishonest peer a node will not change its mind from the original value v	81
5.1	Nodes X and Y exchange digital goods via an FE protocol. Node X sends d_X and receives d_Y and node Y sends d_Y and receives d_X	88
5.2	The Existential Unforgability under Adaptive Chosen Message Attack (EUF-CMA) security game. (1) a key pair, (pk, sk) is generated, (2) and the adversary $\mathcal{A}$ must provide a message m and signature s such that s is a valid signature on m under sk . The adversary is provided with the public key pk and an oracle capable of generating signatures under sk on arbitrary messages OSign_{sk} . (3,4) the adversary wins if they can provide a valid signature and message pair where the message has not been sent to the signing oracle.	91
5.3	SMUS solution space of secure multi-user computation platforms, with the TCM subspace highlighted.	97
5.4	The high-level workflow for programs executed on DSCS. Nodes derive <i>Work Fragments</i> from Programs at either compile-time or run-time, these are then delegated to nodes in the network, who produce results that are combined into an overall result.	99
5.5	A key for the state transitions for DSCS. State transitions colour coded to match whether the state transition was prompted by a <i>Worker Node</i> to <i>Work Issuing Node</i> communication (orange), <i>Work-Issuing Node</i> to <i>Worker Node</i> communication (blue), internal <i>Work-Issuing Node</i> logic (green) or an unconditional transition (black).	101
5.6	DSCS Work Fragment.	102
5.7	Typical DSCS <i>Work Fragments</i> in each of the three <i>Work Fragment</i> states, and the results object returned by a remote <i>Worker Node</i> . This fragment has been approved for remote execution. The single element of OPERATIONS is an ADD and takes as input the 0th and 1st elements of DATA . The operation is popped, executed, writes the output to the second element of DATA . The second element of DATA is flagged as the output of this <i>Work Fragment</i> , here denoted by an extra set of square brackets.	103
5.8	<i>Work-Issuing Node</i> transitions from state Neutral to state Pending Incomplete Work Fragments as a result of an pending <i>Work Fragments</i> not in the COMPLETE state. The same node then transitions to Pending READY Work Fragments as a result of an internal event which indicates there are pending <i>Work Fragments</i> in the READY state.	104

5.9	<i>Work-Issuing Node</i> receives a request from a <i>Worker Node</i> interested to know if there is any pending work, r_a . The <i>Work-Issuing Node</i> replies with s_a , which in this case is NO.	105
5.10	A <i>Work-Issuing Node</i> receives a request, r_j from a <i>Worker Node</i> to join an existing <i>Work Group</i> . The <i>Work-Issuing Node</i> sends a confirmation s_c , and transitions to state Neutral . . .	105
5.11	The <i>Work-Issuing Node</i> transitions from the Pending READY Work Fragments state to S when there are <i>Worker Nodes</i> in the <i>Work Group</i> without an assigned <i>Work Fragment</i> . The <i>Work Fragment</i> is sent to the <i>Worker Node</i> in message s_w to be evaluated by function Exec. The array of result data, r_c , is sent from the <i>Worker Node</i> to the <i>Work-Issuing Node</i> who processes the output and sends confirmation s_c	106
5.12	A <i>Worker Node</i> sends a <i>Work Group</i> removal request r_r to a <i>Work-Issuing Node</i> . The <i>Work-Issuing Node</i> actions the request and sends acknowledgement s_c	106
5.13	Public DSCS <i>Work-Issuing Node</i> interface in the <i>Nosy Nodes</i> threat model.	107
5.14	A state machine for prototypical DSCS work-issuing node behaviours in the <i>Nosy Nodes</i> threat model. All r transitions are messages from remote worker nodes, and all s transitions responses sent from the work-issuing node.	107
5.15	An extension to the mechanism for issuing work given in Figure 5.11 to support a credit reward for completed <i>Work Fragments</i> . The new addition is highlighted in a red square. The <i>Work-Issuing Node</i> keeps an internal counter (initialised to 0) for each collaborating <i>Worker Node</i> . When the result of a <i>Work Fragment</i> is returned, the <i>Work-Issuing Node</i> increases the internal credit counter relevant to that i th <i>Worker Node</i> , c_{wi} , by the value of the <i>Work Fragment</i> , provided by function Cost.	111
5.16	A <i>Worker Node</i> with a pre-existing positive credit balance sends a <i>Work Fragment</i> completion request r_w to the <i>Credit-Issuing Node</i> with which it had accumulated credit. The <i>Credit-Issuing Node</i> evaluates the <i>Work Fragment</i> , decreases the credit total c_w by the appropriate cost, and then returns the results to the requesting <i>Work Fragment</i>	112
5.17	Public DSCS <i>Work-Issuing Node</i> interface in the <i>Lazy Nodes</i> threat model.	113
5.18	A state machine for DSCS <i>Work-Issuing node</i> behaviours in the <i>Lazy Nodes</i> threat model with additional heuristic internal state transitions.	113
5.19	An extension to the mechanism for issuing work given in Figure 5.15 to support the verification of returned results, the new behaviour is highlighted in a red box. The black box VC scheme used here is assumed to have a minimal implementation, such as the VC scheme presented in Chapter 3, but this can be easily made more general with the addition of the standard VC functions for problem generation and evaluation.	118
5.20	DSCS credit generation, given signing secret key sk , <i>Worker Node</i> public key i , previous counter value d , new credit value v_n , and previous credit balance v_b	119
5.21	DSCS credit validation. Given the credit token r_c and corresponding signature r_s returns true if the signature is valid and the embedded counter is greater than equal to the counter on disk.	119
5.22	The process for evaluating a <i>Work Fragment</i> r_w conditional on the validity of credit token signature pair (r_c, r_s) and suitable credit balance.	120
5.23	An extension to the mechanism for issuing work given in Figure 5.15 to support cryptographic tokens attesting to credit balances.	120
5.24	Extension to the mechanism for credit redemption in Figure 5.16 to support a cryptographic token of the credit balance.	121

5.25	Using an FE scheme to mediate the exchange of credit tokens and <i>Work Fragment</i> results. . .	122
5.26	Mediating the exchange of results for an updated credit balance when redeeming credit. . . .	122
5.27	A state machine for DSCS <i>Work-Issuing</i> node behaviours in the <i>Untrusted Nodes</i> threat model, deploying cryptographic tokenised credits and VC, with additional heuristic internal state transitions.	124
5.28	Adversary $\mathcal{B}$; capable of winning the <i>VerifyGame</i> security game as defined in Figure 3.2, violating the <i>Soundness</i> property of an underlying VC scheme given adversary $\mathcal{A}$, capable of producing forged results to computations that evade DSCS detection.	125
5.29	The credit forging security game. A pair of keys are generated for signature scheme <i>SIG</i> , adversary $\mathcal{A}$ is given access to the public key and an oracle able to provide valid credits to their specification. The adversary wins if they are able to provide a valid credit that was not signed by the credit generation oracle.	127
5.30	Adversary $\mathcal{A}$, capable of winning the <i>CreditForgingGame</i> as detailed in Figure 5.29. Given the availability of credit generation oracle, when provided a public key $\mathbf{pk}$ the adversary is able to generate forged credit c	127
5.31	Adversary $\mathcal{B}$; capable of winning the EUF-CMA security game, as given in Figure 5.2, given the existence of adversary $\mathcal{A}$ from Figure 5.30. Queries $\mathcal{A}$ makes to the credit generation oracle are passed through to the signing oracle available to $\mathcal{B}$. The forged credit output by $\mathcal{A}$ is passed through as the forged signature output by $\mathcal{B}$	127
5.32	A map of the possible speed ups for a given program depending on the ratio of parallel work to sequential work, $c_p : c_s$, and the number of available nodes n . Speed ups are given as a multiplicative factor over executing the target computation with a single <i>Worker Node</i>	130
5.33	The theoretical latency of a computational task delegated to the DSCS network.	131
5.34	Theoretical DSCS Levelled Fully Homomorphic Encryption (LFHE) ciphertext evaluation latency t_{HEDSCS} as compared with local-only plaintext execution latency t_{local} and local-only LFHE ciphertext evaluation latency t_{HE} , assuming no VC or VC that does not inhibit parallelism.	131
6.1	Public DSCS <i>Work-Issuing Node</i> interface in the <i>Nosy Nodes</i> threat model, duplicated from Figure 5.13, annotated with the implementation target functions.	139
6.2	DSCS Work Fragment, duplicated from Figure 5.6.	139
6.3	Results object structure, duplicated from Subfigure 5.7c	139
6.4	BFV cryptosystem function definitions adapted from [52]. Where N is the coefficient modulus for the space of ciphertexts, t is the coefficient modulus for the space of plaintexts, M is an expansion factor for the relinearisation procedure, and $\Delta = \lfloor N/t \rfloor$	141
6.5	High-level DiSCuS program structure.	145
6.6	DiSCuS Program Processing	147
6.7	A DiSCuS program for adding two pairs of integers.	148
6.8	The <i>GlobalInstructions</i> generated by processing the program defined in Figure 6.7. The instructions in (a) are related to their counterparts in (b) by red arrows.	149
6.9	Homomorphic preprocessing, expanding <i>GlobalInstructions</i> into <i>GlobalInstructions</i> with ciphertexts expressed as <i>CipherTextFragments</i>	150
6.10	Consolidating <i>GlobalInstructions</i> with ciphertexts expressed as <i>CipherTextFragments</i> into <i>WorkFragments</i>	151

6.11 Results of outsourced WorkFragment evaluation.	152
6.12 Duplicate of Figure 5.33. t_{DSCS} is the target performance for DiSCuS over baseline t_{local} benchmarks.	152
6.13 The SIMD lanes available to a BFV plaintext with ring order 16 for polynomial addition and polynomial multiplication. Each position, m_i , conceptually aligns with a polynomial coefficient of order $i - 1$	154
6.14 Experiment 1: Latency of 1,000 independent plaintext additions on the DiSCuS network. The line labelled c provides context of the local (interpreted) execution latency. A dotted line, t_{DSCS} , shows the predicted performance given overhead constants, defined in Table 6.4. The prediction t_{DSCS} has an R^2 value of 0.78.	155
6.15 Latency of 1600 equivalent-plaintext-addition-operations via independent homomorphic additions on the DiSCuS network for both conventional ciphertext representation and DCRT ciphertext representation.	156
6.16 Latency of 1600 equivalent plaintext addition operations via dependent homomorphic additions on the DiSCuS network for both conventional ciphertext representation and DCRT ciphertext representation.	157

List of Definitions

2.2.1 Definition (Network Identity)	9
2.2.2 Definition (Node)	9
2.2.3 Definition (Swarm)	9
2.2.4 Definition (Central Authority)	9
2.4.1 Definition (IND-CPA Secure Encryption Scheme)	15
3.2.1 Definition (Completeness)	23
3.2.2 Definition (Soundness)	24
3.2.3 Definition (Outsourceable)	25
3.2.4 Definition (Prover Computational Cost)	25
3.2.5 Definition (Verifier Computational Cost)	25
3.2.6 Definition (FactorGame Advantage)	26
4.2.1 Definition (Completeness)	51
4.2.2 Definition (Strong Fairness)	52
4.2.3 Definition (Symmetric IND-CPA Secure Authenticated Encryption Scheme)	53
4.4.1 Definition (Computational Completeness)	55
4.4.2 Definition (Computational Strong Fairness)	56
4.8.1 Definition	67
4.8.2 Definition (Repeat Round Fairness)	69
4.9.1 Definition	74
4.11.1 Definition (Termination)	79
4.11.2 Definition (Agreement)	79
4.11.3 Definition (Validity)	80
5.3.1 Definition (EUF-CMA Secure Signature Scheme)	91
5.3.2 Definition (Threat Model: Nosy Nodes)	91
5.3.3 Definition (Threat Model: Lazy Nodes)	91
5.3.4 Definition (Threat Model: Untrusted Nodes)	92

5.4.1 Definition (Multi-User Computation Platform)	93
5.4.2 Definition (Data Availability)	93
5.4.3 Definition (Data Confidentiality)	93
5.4.4 Definition (Data Integrity)	93
5.4.5 Definition (Secure Multi User Systems (SMUS))	93
5.5.1 Definition (Distributed SMUS)	95
5.5.2 Definition (Decentralised SMUS)	95
5.5.3 Definition (Fair Resource Allocation)	95
5.5.4 Definition (Low Longevity Cost)	96
5.5.5 Definition (Trusted Computation Mesh (TCM))	96
5.6.1 Definition (Work-Issuing Node)	100
5.6.2 Definition (Work Group)	100
5.6.3 Definition (Worker Node)	100

List of Acronyms

BOINC Berkeley Open Infrastructure for Network Computing

DCRT Double Chinese Remainder Transform

DLP Discrete Logarithm Problem

DSCS Distributed Secure Computation System

EUF-CMA Existential Unforgability under Adaptive Chosen Message Attack

FE Fair Exchange

FHE Fully Homomorphic Encryption

FaaS Functions as a Service

GCD Greatest Common Divisor

HE Homomorphic Encryption

IND-CCA Indistinguishability under Chosen Ciphertext Attack

IND-CPA Indistinguishability under Chosen Plaintext Attack

IOP Interactive Oracle Proof

IoT Internet of Things

LFHE Levelled Fully Homomorphic Encryption

LWE Learning With Errors

MAC Message Authentication Code

mDNS Multi-cast DNS

MPC Multi-Party Computation

NTT Number Theoretic Transform

PCP Probabilistically Checkable Proof

RLWE Ring Learning With Errors

RNS Residue Number System

SIMD Single Instruction Multiple Data

SMUS Secure Multi User System

TCM Trusted Computation Mesh

TPM Trusted Platform Module

VC Verifiable Computation

ZKP Zero Knowledge Proof

Chapter 1

Introduction

Consider a smart home, with dozens of devices spread throughout, each performing some small infrequent computational task. Homes are full of underutilised personal computers, tablets, smart phones, smart watches, and Internet of Things (IoT) devices such as smart fridges, and intelligent thermostats. Between short bursts of use most of these devices sit idle with their capability wasted. The prevalence of these idle devices raises the question, might it be possible to network these devices together to cooperate on joint computational tasks? This would allow devices to assist in the computational tasks of their peers in order to gain some utility between calls to their primary functions, and overall computational tasks would benefit from reduced latency. The opportunity to leverage all this unused computational power is the primary motivation for the work in this thesis.

Such a decentralised distributed computation network, if not designed carefully, would be a tantalising target for a whole host of malicious actors: IoT devices wishing to burn out the batteries of competitors, an economically incentivised cryptocurrency mining operation, or perhaps a botnet master wishing to grow their swarm exponentially. The importance of designing a network like this with a security-first mindset is not to be understated.

Let's further dissect some potential malicious actions against a decentralised distributed computation network. Consider first a malicious actor who wishes to disrupt computations on the network. To this end such a malicious actor may corrupt the results of computations delegated to them. These corrupted results could see program executions yield incorrect results or fail outright. A second class of malicious actor may simply behave lazily in the network, offloading computation to their peers but not offering up their computational capacity in kind. A third consideration when designing a distributed computation protocol is how to handle private data. If programs must be carefully crafted in order to avoid leaking any personal information the space of possible applications shrinks massively.

The new abstract class of protocols introduced in this work is called a Trusted Computation Mesh (TCM). In relation to the discussion above, TCM is a decentralised distributed computation protocol in which results cannot be falsified, lazy engagement with the network is suboptimal behaviour, and ensures the confidentiality of private data. The work in this thesis culminates in a concrete TCM protocol, called Distributed Secure Computation System (DSCS).

In order to guarantee the correctness of the results of outsourced computational tasks, DSCS employs Verifiable Computation (VC) schemes. VC schemes are a class of cryptographic construction used to verify the results of outsourced computations. In order to deter lazy interactions with the network, DSCS deploys

a novel credit system. This was designed to have as minimal a data footprint as possible as to not add unnecessary overhead to the system. A secondary concern is the fair and equitable exchange of results for credit. This is solved with the integration of Fair Exchange (FE), which is used in DSCS to support the atomic exchange of credit for computational results. FE protocols try to provide a mechanism by which two parties can exchange digital goods without either opening themselves to the possibility of being cheated out of the other parties secret. It is widely believed that FE is impossible without involving a trusted third party. The widely sighted impossibility result only holds for a subset of the FE problem, namely deterministic FE protocols in asynchronous networks. To the best of the authors knowledge this thesis includes the first two party FE scheme. Proving the construction of a two party FE scheme is possible and overturning a widely held belief in modern cryptography. The third TCM property, private data confidentiality, is guaranteed in DSCS with two techniques, the first of which is variable poisoning. Private data, or any data derived from private data can be appropriately flagged and these instructions executed locally. The second technique used to maintain private data confidentiality is Fully Homomorphic Encryption (FHE), or more precisely a more restricted but performant variation Levelled Fully Homomorphic Encryption (LFHE), which is used to mask private data without restricting the computational substrate to trusted entities. LFHE is a natural task for a distributed computation network as popular acceleration techniques for lattice cryptosystems introduce trivial parallelism to programs. The computational burden of FHE can therefore be easily shared amongst collaborators the network.

1.1 Motivations

In order to be functional, devices tend to be designed for their expected maximum workloads. Instead, devices could be designed to handle their expected average load, and leverage all idle and unused computational power spread throughout the home to assist with burst performance? The trend seen since the introduction of the personal computer is the substitution of monolithic single systems for a diverse array of smaller specialised devices. Yet, there exists no easy mechanism for these devices to collaborate on computational tasks with each other.

One of the primary motivations for this work are recent developments in LFHE scheme design and LFHE acceleration. Improvements here open a potential opportunity for outsourcing programs that operate on private data to untrusted workers. In short, LFHE schemes allows data to be processed while encrypted. A standard suite of arithmetic operations can be applied to ciphertexts with predictable results on the underlying plaintexts. The value of applying LFHE to the IoT setting is clear: Sensitive personal information harvested from sensors embedded in the home can be encrypted and processed on many devices without fear of sensitive information leaking to nosy participants. Additionally, devices become resistant to leaking data via device theft, as the data is encrypted while at rest and in flight.

Historically FHE has been out of the reach of all but the most powerful computers, as such it has often been characterised as more of a curiosity than a viable tool. Recent work involving FHE algorithmic development and acceleration have argued this is no longer so. But, despite massive improvements in ciphertext evaluation it is still massively more demanding than plaintext execution. As such FHE is still outside the capabilities of severely restricted computational hardware. Part of the work in this thesis aims to partition and distribute homomorphic evaluation across a swarm of cooperating nodes so that FHE workloads can be executed on severely computationally restricted hardware. Much of the work in this thesis follows from the critical observation that the primary LFHE acceleration techniques introduce trivial parallelism to FHE evaluation,

even in fully sequential programs. A consequence of this observation is that the operational cost of FHE could conceivably be absorbed by distributing the workload across a network of previously-idle devices.

This work is made possible because of the following key advances:

- Modern performant LFHE schemes, which require less compute than FHE but with some computational restrictions.
- High levels of parallelism in modern lattice based LFHE schemes when implemented via Residue Number Systems (RNSs).
- Increasing numbers of low-powered personal computation devices which wish to compute on potentially-sensitive user data.

Together these show not only the need for a system supporting secure decentralised distributed collaborative computation with secure-by-default data management, but also that many of the tools to build such a system exist.

1.1.1 Applications

A system supporting decentralised secure computation has a diverse set of potential applications, which will now be briefly explored. All of these share a common context: the deployment of completely trusted or semi-trusted devices where a protocol could support organic load balancing and collaboration.

A TCM, such as the one presented in this work, could be deployed to great effect anywhere a lack of a central authority, power restrictions, or data sensitivity restrictions, would normally not allow devices to collectively operate on data. For example, distributed sensors operating in areas with reduced access to the internet, such as a field of crops with sensors. In such a setting resisting data leaks through physical device theft is an operational priority. An existing technique would be to encrypt harvested data with a public key encryption scheme, such that the data can be decrypted and processed at a later date. By substituting the public key encryption scheme for a scheme with LFHE properties devices are not only secure against data leakage, but devices can also begin to preemptively process the data, using the TCM to share the computational load across the network.

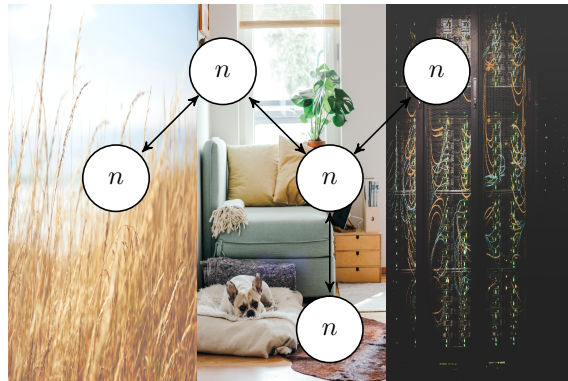


Figure 1.1: Distributed computation protocol motivating example: sensors in a field, IoT devices in a home, and a data centre deployment.

Another compelling application for a TCM is in the context of IoT devices in the home. With the addition of a TCM devices would be able to perform data processing operations while conscious of the sensitivity of

the data being processed in a wide, and often insecure [53], ecosystem of IoT device providers. Data can be encrypted at rest and in-flight and devices can load-balance the computational burden between themselves with no need for third party trust between devices from different vendors. Figure 1.1 shows an abstract network of collaborating nodes in a selection of motivating domains.

An added benefit of distributed systems, especially ones designed to work with a variable and ad hoc network topology, is their resilience against individual device failure. This feature is particularly useful in a setting where providing regular maintenance is impossible. A set of small satellites with a unified purpose would be able to adaptively reorganise the execution of programs depending on the availability of nodes. One could also imagine a data centre deployment where the server machines can dynamically self-organise to maximise computational throughput while maintaining secure data boundaries between applications and resisting service outages due to individual node failures.

A successful TCM implementation would provide the ability for individual devices to share the burden of workload spikes. This could potentially encourage a design philosophy which champions devices with adequate computational resource to handle their expected average workload, with the knowledge that a network of devices can be leveraged to assist with their maximum load.

The focus of this thesis is not to develop specific applications. Instead, general mechanisms are supplied that can be applied to a variety of applications. These techniques are motivated by their relationship to secure decentralised distributed computation but in fact the individual mechanisms detailed in this work have wide ranging applications. In the chapters detailing specific contributions, the applications both within and beyond distributed computation are explored.

The general methodology proposed in this thesis, that of a TCM, attempts to solve as generic a case as possible. This is evident from the wide ranging application space in the discussion above. However, when considering IoT in particular some of the security assumptions present in this work can be relaxed. For example, many IoT systems have a centralised setup which could be used to provide a concrete defence against wanton identity regeneration - an attack our proposed TCM only provides a heuristic defence for. We consider IoT a special-case of the general application space targeted and leave IoT-specific optimisations to future work.

1.2 Motivating Adversarial Behaviours

By considering potential adversarial behaviour of nodes in the network definitions for the expected soundness properties of a TCM can begin to take shape. The approaches taken in DSCS to defend against these types of adversarial activity are summarised in this section.

1.2.1 Corrupting Computation

An adversary may be motivated to tamper with the results of computations by providing incorrect results for any program fragments that are delegated to them. The incorrect results can be random data sent to corrupt the overall results of a program, or instead it could be specifically crafted to alter the results in a way beneficial to the adversary. By way of a defence against this DSCS uses VC protocols which are a well studied area of cryptography, designed to ensure provided results are the correct output of computation. VC schemes require that the ability of an adversary to convince the originator of a program of an incorrect result reduces to some computationally intractable problem. Chapter 3 details a new VC protocol with a number of novel properties making it well suited for distributed computation.

1.2.2 Avoiding Work

A challenge of designing a secure distributed computation mesh involves ensuring nodes proactively engage in the computations suggested by their peers. In an IoT setting proactive behaviours will result in improved perceived performance of their peers, who may originate from competing brands, at the cost of their own effort. Engaging in the protocol burns through a device's battery, and gives their competition an advantage. To encourage engagement in the network DSCS includes a minimal tit-for-tat credit scheme, in which work done now on behalf of another node can be exchanged for work done later on the remote node. In this way any lazy nodes are actively encouraged to proactively engage with the network, as any work they do now is offset by the computational power they can leverage later. The mechanics of this credit scheme are included as part of the description of DSCS in Chapter 5.

1.2.3 Unfair Exchange

With the addition of a credit scheme another problem is introduced, that of the fair exchange of digital goods. If a node is completing a large volume of work for a peer they may be uncomfortable releasing the results until they hold the credit. Mirroring this, a node expecting the result of a large operation might be uncomfortable with the idea of releasing any credit before receiving the results. This deadlock is a classic problem in the exchange of digital goods. FE schemes are a class of cryptographic protocols designed to provide assurances to both parties and as such swap the output of computations for credit. A first-of-its-kind FE scheme is presented in Chapter 4 which supports the fair exchange of digital goods between two parties without the need for a trusted third party.

1.2.4 Viewing Private Data

The most immediate concern for any user of a distributed computation network should be the potential to leak private data as work is farmed out to untrusted or malicious nodes. The mitigations against this are twofold. First, DSCS allows programs to poison data with a flag indicating it is private. Any variables derived from private data are similarly poisoned. The data flagged as private never leaves the host node. Second, DSCS is able to leverage a LFHE scheme and encrypt data before processing. The details of data poisoning and LFHE integration are specified in Chapter 5 as part of the DSCS protocol specification.

1.3 Challenges

There are a number of research challenges that need to be solved to satisfy the objectives listed above. First is that of the computational cost of existing VC schemes. The existing computational models do not track well to a system that cares about how much work a remote worker node has to do to produce a result that is easily verified. In a distributed computation setting an ideal VC protocol would not dramatically effect the computational effort required by all participants. New work must be completed that prioritises minimising the cost of VC protocol engagement for all parties.

Currently there exists no FE protocols that do not require the involvement of a trusted third party at some stage. The state-of-the-art only requires the third party to intervene in the case of a dispute [79], but in a decentralised distributed computation protocol there are no central authorities and therefore there can be no trusted third parties. A widely-cited impossibility result claims an FE protocol between only two parties is impossible [93]. Considerable work must be completed to fully explore the implications of this impossibility

result to devise any edge cases in which two party FE may be possible. In order to leverage FE in this work a FE scheme for only two parties must be constructed.

It is not clear how one would construct a secure distributed computation protocol. In order to construct a protocol of this type work must be done to fully conceptualise the threat vectors against such a system and design appropriate mitigations. These qualities need to be properly defined in order to allow concrete proofs to be constructed relating to various notions of soundness. Beyond the abstract definitions of a secure distributed computation protocol, a number of smaller sub-problems must also be solved. These problems relate to various areas, such as: designing how workloads will be distributed, how cryptosystem primitives will be integrated, and interfaces exposed to other nodes in the network. From the onset it is not clear how a distributed network such as this will perform, and even if it is worth investing the computational overhead required to engage in such a network over isolated computation.

1.4 Contributions

The high level objectives of this thesis are to:

1. Ensure the correctness of outsourced computations, with better-than-state-of-the-art computational complexity with single-use programs.
2. Provide apparatus for two mutually distrusting parties to fairly exchange digital information without a trusted third party intermediary.
3. Define the set of behaviours and security considerations for a platform supporting ephemeral decentralised distributed computation.
4. Demonstrate the efficacy of an ephemeral decentralised distributed computation platform, bound the engineering targets required for a compelling implementation, and guide further development work in this area.

These objectives are reflected in the contributions of this PhD:

1. A novel verifiable computation scheme, **Proofless Verifiable Computation from Integer Factoring**, inspired by common acceleration techniques for arithmetic over large integers. Although the Verifier overhead is relatively large compared to the existing schemes, it is a constant fraction of the overall computation. This means that unlike the related work, it is always viable. Furthermore, the scheme has a minimal fixed time set up, has zero program preprocessing, has zero sized proofs, and has an optimal workload for the remote worker. These are properties not present in the related work.
2. **Two Party Fair Exchange**, the first two-party fair exchange protocol. To the best of my knowledge this is the first example of a FE protocol that does not require a trusted third party arbitrator at any stage. This result overturns a widely cited impossibility result, a close reading of which suggests the proof only holds for a deterministic program in a fully asynchronous network.
3. Definitions for **DSCS**, a candidate TCM. Provided alongside definitions and proofs to support the claim that it satisfies the properties of a TCM. Some additional minor contributions are included in the definition of DSCS. These include:
 - Definitions for the TCM, a class of secure decentralised distributed computation platforms

- Mechanisms for a small-footprint point-to-point credit system.
 - Deep integrations for optimising cryptographic primitives in a TCM.
 - Easily extensible with many advanced cryptographic constructions.
4. **DiSCuS**, a programming framework and DSCS reference implementation written in Rust. It is used to acquire real-world metrics about DSCS performance. The framework includes an interpreter for a DSCS-specific instruction set, as well as the daemon used to communicate with the wider distributed network, delegate work, and consolidate results.

1.5 Publications

The work in this thesis is derived from work from the following preprint publications:

- A. Dalton, D. Thomas, and P. Cheung, “A Novel Preprocessing-Free Proofless Verifiable Computation Scheme from Integer Factoring”, Cryptology ePrint Archive, 2023 [48].
- A. Dalton, D. Thomas, and P. Cheung, “Two Party Fair Exchange”, Cryptology ePrint Archive, 2023 [49].
- A. Dalton, D. Thomas, and P. Cheung “DiSCuS: Towards Distributed Secure Computation”, as yet unavailable.

1.6 Thesis Structure

The next chapter, Chapter 2, includes general background information relevant to most of the works in this thesis. This covers distributed computation protocols, existing approaches to the adversarial problems considered above, RNSs, a primer on cryptographic primitives and proof conventions, and an overview of the notation used throughout this thesis.

After the Background, the main content chapters are organised to have a self contained introduction, background, related work, content, evaluation, and conclusion sections specific to the contributions of that chapter. The content chapters correspond to the four main contributions to the work:

- Chapter 3: Proofless Verifiable Computation from Integer Factoring.
- Chapter 4: Two Party Fair Exchange.
- Chapter 5: DSCS: A Trusted Computation Mesh.
- Chapter 6: DiSCuS: A DSCS Implementation.

There is some interdependence between the later chapters, but most of the relevant concepts are reiterated where necessary. DSCS makes black box use of the work on VC in Chapter 3 and also the work on FE in Chapter 4, so an understanding of the interfaces and properties are important. DiSCuS, in Chapter 6, by virtue of it being a DSCS implementation, relies heavily on the work in Chapter 5. The dependencies between the chapters are shown in Figure 1.2.

Following these core chapters this thesis finishes with a overall conclusion in Chapter 7.

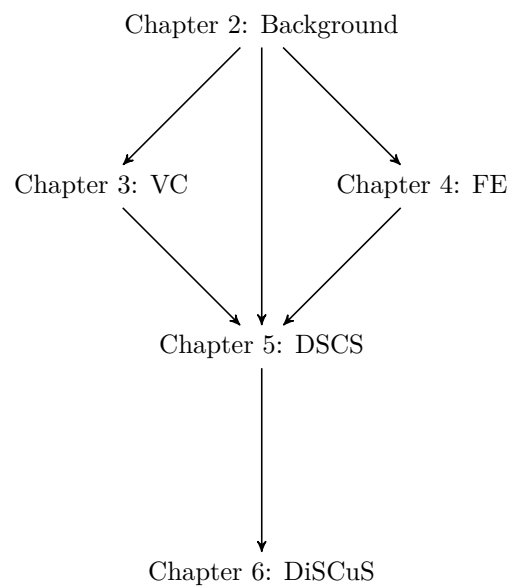


Figure 1.2: Dependencies between the background and content chapters in this thesis, with $A \rightarrow B$ indicating Chapter B builds on the content of Chapter A .

Chapter 2

Background

This chapter sets out the general background used throughout this work with the information relevant to multiple subsequent chapters presented here. Alongside the shared background in this chapter, each subsequent content chapter has their own section providing the background context specific to that chapter.

2.1 Notation

Unless otherwise specified, capital letters are used to represent sets, with the size of a set A given by $|A|$. There are a few cases where capital letters do not refer to sets, as to better align this work with conventions in related fields. The first exception is for the large composite integer N , as is usual for RNS work. Similarly, in general the letter R is reserved to represent a ring, with the exception of $\mathbb{Z}_x$ which is the ring of integers modulo x . More domain-specific exceptions are introduced in relevant chapters.

Generic functions are given the letter f , whereas named functions are written in a specific **Function** font and are usually provided alongside a corresponding definition. In function signatures $\leftarrow$ denotes output, such as $y \leftarrow f(x)$, and in pseudo-code descriptions it indicates an assignment. $\leftarrow \$$ is used where a random element should be sampled from a domain, such as $r \leftarrow \$ X$. The symbols $\equiv$, and $=$ represent congruence and equality respectively. The use of $\equiv$ implies a modular reduction where one is not explicit.

2.2 Distributed Networks

Distributed networks form a general backdrop for much of the work presented as part of this PhD. When discussing distributed networks the following disambiguated definitions are used:

Definition 2.2.1 (Network Identity). An individually distinguishable cryptographic identity, usually associated bijectively with a user.

Definition 2.2.2 (Node). An independent compute resource, connected to other nodes in a network.

Definition 2.2.3 (Swarm). A set of nodes, networked together, engaged in some distributed protocol.

Definition 2.2.4 (Central Authority). A protocol participant with elevated privileges, able to provide a canonical source of truth.

Distributed protocols allow for inter-device communication and cooperation, often without the oversight of a central authority. Networks designed without a central authority are known as *Decentralised*. Within distributed networks, nodes are organised in a non-uniform topology. Communication with neighbours can be intermittent, and often there are no assumptions made about node availability.

One of the most widely known examples of a distributed network is BitTorrent [42], which is a protocol providing efficient and resilient file transfers over a network. Interconnected nodes in a BitTorrent network share fragments of files as they are downloaded, dramatically reducing bandwidth bottlenecks and distributing the bandwidth required to transfer files to interested nodes between all nodes participating in the download. As more devices share files, downloading files becomes quicker, easier, and more resistant to failure caused by single node outages.

BitTorrent has a variety of mechanisms to reduce the impact of misbehaving nodes. One such mechanism is “tit-for-tat” exchanges, in which data is sent to peers who first send data in return. This discourages lazy users, who otherwise would be motivated to reduce their communication load by receiving network data without offering any data up. This has the unfortunate side effect that new nodes are unable to access any files as they have no files to offer up themselves. A technique called “optimistic unchoking” reduces these effects by having nodes reserve some of their bandwidth for sending files to nodes that would usually be ignored due to the “tit-for-tat” exchange policy. There is a non-negligible possibility that a file offered up for sharing becomes stale and inactive, reducing its availability to the swarm. The individual cost to a node to maintain the availability of an unpopular file it is hosting is much higher than with a more popular file, due to the shared load between various nodes. These policies inform some of the choices made in the design of the distributed computation protocol in Chapter 5. Despite the availability of these policies BitTorrent ultimately relies on the altruistic behaviour of the majority of nodes to propagate files and maintain file availability.

Another example of a distributed network protocol is Onion routing [62], which supports decentralised and anonymous internet traffic. This is achieved with layers of node-specific encryption which only divulge enough information for the network packet to reach the next node in the sequence. No nodes along the route know any information about the origin node and destination node of the data other than the origin and the destination. Although much slower than traditional packet routing it provides a dramatic improvement in both client and server anonymity.

One common issue for a decentralised network is migrating DNS resolution to a mechanism that doesn’t require an authoritative domain name server. To that end Multi-cast DNS (mDNS) [38] is used to provide decentralised host-name to IP address resolution. In mDNS, a node that wishes to connect a host-name to an IP address broadcasts a request to the network, then the node with a matching host-name broadcasts a response, allowing all network nodes to keep an up to date copy of all network IP addresses. The mDNS protocol is a common building block when constructing a distributed protocol. mDNS is used in this work as an off-the-shelf solution, used when nodes connect to the swarm in order to provide them with a mechanism by which to route traffic to other nodes.

Cryptocurrency is a widely known example of another distributed and decentralised system, built on a blockchain protocol, wherein mutually untrusting nodes can reach consensus on a sequence of transactions applied to a database. In this way a blockchain protocol provides a decentralised database governed by network consensus completely devoid of a central authority. This database technology has contributed to an eruption in decentralised e-currencies such as Bitcoin [89] and Ethereum [32], which both use a blockchain database to record the number of coins held by network participants. In order to transfer currency, a change

to the database must be recorded, which requires the network to append a block to the chain of blocks listing all database changes. This is a computationally hard task with a difficulty dynamically chosen to govern the interval between new blocks. As the number of nodes attempting to append a block to the network grows, the difficulty of appending a block becomes artificially harder to ensure the interval between blocks is consistent. Nodes in the network search for the next block, an activity colloquially known as “mining”, and the probability of discovering it is governed by the computational power they can muster. This proof-of-work mechanism leads to a one-CPU-per-vote style of consensus secure against Sybil-style attacks [50], where a malicious actor generates a host of valid accounts to overwhelm a voting population. The nature of proof-of-work consensus produces a massive computational burden on participating nodes, which has resulted in massive amounts of power used globally to support blockchain-backed cryptocurrencies [92]. There have been some efforts to improve this by moving to a proof-of-stake mechanism for consensus, which relies on users “staking” financial assets on-chain, and any dishonest behaviour results in a forfeiture of posted assets [33, 77]. Participants are selected to validate a block of transactions with their probability of selection proportional to the value of their posted assets. Blockchain-based solutions offer a fantastic solution for trustless database management, but the inefficient use of network-attached compute power and the constraints on smart contracts makes them ill-suited for untrusted distributed collaborative computation. Blockchains inherently require some restricted resource to drive a proof-of-X mechanism of consensus, this overhead is untenable in this setting, and it seems unlikely that the strong consensus provided by the blockchain is required.

2.2.1 Distributed Computation

Distributed computation networks are a class of 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 [16], ParaWeb [30], ATLAS [15], and BOINC [9]. These all provide the component building blocks to construct a distributed computational platform, such as; isolated execution environments, program specification, and network primitives. These only support distributed computation in centralised networks, with vary degrees of protections against malicious participants.

The existing frameworks have been used to build operational distributed computation projects [9, 18]. Due to the constraints on the security model imposed by the protocols and frameworks underpinning distributed computation projects, namely centralisation, they are usually constructed to tackle problems in the domain of scientific computation. In these volunteer distributed scientific computation projects, participants from around the world donate spare compute resource for the public good. One of the most well known examples is Folding@home [18], in which participants perform protein folding simulations to advance research in the field of organic chemistry. In the wake of the COVID19 health crisis [41] Folding@home saw an influx of volunteers and at its peak was able to leverage 1.01 exaFLOPS of donated computational power [115]. To reward volunteers Folding@home uses gamification, where volunteers are rewarded for computing results with points and a position in a league table.

The frameworks underpinning Folding@home and other volunteer distributed computation projects share a similar threat model. The threat model assumes that because users are willing to donate the power of their computer to the project they will be acting altruistically. There is some mention in these works about the threat of executing arbitrary code on a volunteer’s computer, a threat mitigated by proper sand-boxing such as in ParaWeb where the Java Virtual Machine runtime environment is leveraged to this effect [30]. The great deal of effort that has gone into securing the machines of any volunteers, presumably to increase

their trust in the computation platform, contrasts with little consideration afforded to the inverted threat situation, in which volunteers may try to falsify results. In this scenario a volunteer wishes to not do the work delegated to them and instead send falsified results that give them the reward for a task well done without expending any of the requisite effort. Unfortunately where something is gamified, as protein folding is with Folding@home, it is conceivable that people will cheat to climb the rankings. Couple this with the fact that Folding@home encourages teams to participate together, they must be open to the possibility of falsified results and collusion amongst users. There are some weaker protections against this adversarial model, primarily to defend against accidental errors where a volunteer unwittingly return incorrect results. In order to defend against these accidents work is duplicated until a result is corroborated by an additional node, this is an obvious opening for collusion.

To the best of my knowledge there are no active distributed computation projects which allow work to be done by arbitrary anonymous participants outside of the field of volunteer scientific computation. This indicates that for the existing security models to work, users need to approach the projects as altruistic volunteers with no motivating self-interest in the work. Consider a more open decentralised distributed computation platform, with little in the way of work validation or mechanisms to enforce proactive behaviour in nodes. Individual node security is a solved problem: code should execute in appropriate sandboxes. However, what is to stop an enterprising individual on the network from creating a distributed workload that does nothing other than mine for cryptocurrency and thereby generate profit? This is why these networks currently operate on tasks originating from a centralised trusted source.

As it is, Folding@home and similar projects work well because, by the very nature of the project, users approach it as volunteers. There is no financial incentive to cheat. At most someone could cheat for bragging rights, but still at fairly substantial personal expense. As work is duplicated, the best an independent node could reasonably do without organising a cabal of cooperators is to be the single incorrect result in a trio of results, the other two of which are correct and sourced from honest users. Furthermore, work duplication is clearly wasteful. Each program has to be executed at least twice to see if consensus was reached; more if a dispute has to be resolved.

Ethereum and similarly styled cryptocurrencies also support distributed computation in the form of *smart contracts*. Smart contracts are short code snippets stored on the blockchain with attached Ethereum currency, which in this context is known as “gas”. In exchange for executing the code, participants in the network get rewarded with this currency. Currently smart contracts can be used to create static code snippets with a strong assurances of being executed given certain triggers, ideal for escrow holdings [60] and cryptographically-backed gambling [56]. This blockchain-stored program logic is triggered by some external signal, in the form of a transaction on the blockchain. These are discussed in more detail later in this thesis as they become relevant.

2.3 Residue Number Systems

For a set of co-prime integers P_N and large composite number $N = \prod P_N$, an RNS [107] is a system for representing a very large integers less than N , as a vector of smaller integers, where each element of the vector corresponds to the target integer reduced modulo a distinct element of the set P_N . It serves as the inspiration behind the VC scheme in Chapter 3 and it enables the acceleration technique in the definition of DSCS in Chapter 5.

The process of converting an integer to a given RNS representation involves reducing that integer by each

of the elements of P_N . For integer x , an RNS representation $X = \{x_1, x_2, \dots\}$ is produced via:

$$X = \{x \bmod p_i \mid p_i \in P_N\} \quad (2.1)$$

Due to the Chinese remainder theorem [43] each integer smaller than N has a unique representation in any RNS. Given the ability to generate multiplicative inverses in the ring R_{p_i} , such that $x(x^{-1}) = 1 \bmod p_i$, the process of reversing an RNS representation to generate the integer $\bmod N$ is easy. If p_i is prime, R_{p_i} is a finite field and therefore each element has a multiplicative inverse and Euclid's algorithm can be used to efficiently invert elements in the ring. A set of intermediate values $\hat{N}$ can be precalculated when the RNS is first instantiated, meaning there are relatively few operations required to convert a number from an RNS form into a conventional representation. The process is as follows:

$$\hat{N} \leftarrow \left\{ \left(\prod_{p_i \in P_N} p_i \right)^{-1} \bmod p_i \mid p_i \in P_N \right\} \quad (2.2)$$

$$x \leftarrow \sum_{x_i \in X} \hat{N}_i \cdot x_i \bmod N \quad (2.3)$$

The central benefit of representing numbers this way is that basic arithmetic modulo N can be performed on each element of the RNS representation independently instead of operating on the very large integer representation. This distinction massively reduces the required word length for operations and allows arbitrarily precise computations to be executed with ease on fixed precision hardware.

$$x \rightarrow X = \{x \bmod p_i \mid p_i \in P\} \quad (2.4)$$

$$y \rightarrow Y = \{y \bmod p_i \mid p_i \in P\} \quad (2.5)$$

$$x + y = \{x_i + y_i \mid x_i \in X, y_i \in Y\} \quad (2.6)$$

It's common best practice when performing calculations modulo a very large value N to accelerate the computation by first converting the operands into RNS form [46]. Along with reducing the required word length of operations to more conventional lengths, such as 64-bit or 32-bit, these schemes have the added benefit of introducing trivial $|P_N|$ -way parallelism into a sequence of operations. Many cryptographic applications, especially FHE, use modulo arithmetic with an extremely large modulus. As such, it is a standard acceleration technique for cryptographic implementation.

2.4 Cryptography

Standard cryptographic notation is used throughout [106]; pk and sk for any public and secret keys, k is used for the key of any symmetric schemes, and λ is the security parameter. $\text{negl}(\lambda)$ is any function that grows negligibly in λ . The special font $\mathcal{A}$ is used to identify an adversarial algorithm. When dealing with concrete security, a scheme is (t, ε) -secure if an adversary bound in time t has, at most, probability ε of breaking the scheme.

Throughout this work a number of common cryptographic constructions, mostly encryption schemes, VC, and FE, are used as black boxes. The interfaces to these schemes are described when relevant but mostly the high level capabilities of these schemes are the focus. The rationale is that by abstracting these classes of cryptographic construction, any higher level constructions produced within this work will survive beyond the discovery of any insecurities in any one specific cryptographic building block.

As with similar cryptographic works, the protocols presented in this thesis are largely set in the *computational security* domain. In contrast to the domain of *statistical security*, in which schemes are reasoned about under attack from computationally unbounded adversaries, *computational security* features adversaries with computational ability which is polynomially bounded in a security parameter λ . When reasoning about these adversaries it is often useful to think in terms of the *advantage* of the adversary, which is taken to mean the probability of an adversary successfully circumventing a chosen security property of a system. When dealing with adversarial advantage, the challenge posed to the adversary is often formulated as a *security game* in which the adversary is issued a challenge they must win, and winning is equivalent to breaking the security of a cryptographic protocol. The adversary may or may not have access to oracles that can be queried to learn additional information about the system, such as providing valid encryptions of plaintext messages provided by the adversary. Examples of security games applied to adversarial advantage are provided in the following paragraphs when describing properties about existing cryptographic constructions.

2.4.1 Encryption Schemes

An encryption scheme $\mathcal{E}$ is defined as the tuple of functions $\mathcal{E} = (\text{KGen}, \text{Enc}, \text{Dec})$, with slightly different function signatures for both private and public key encryption. This work makes use of both private and public key encryption when constructing higher order cryptographic protocols.

Private Key Encryption

An encryption scheme is “private key”, or “symmetric”, if the same key is used for both encryption and decryption. A key in a symmetric encryption schemes is denoted with k . Private key schemes tend to be much more performant than their public key counterparts. When sending messages with a private key scheme there is the obvious issue of key distribution. Symmetric schemes require that both participants already have access to the key or are otherwise able to negotiate a key through a shared secure channel. The terms “private key encryption” and “symmetric encryption” are used interchangeably. Formally, the definition of a private key encryption scheme, $\mathcal{E} = (\text{KGen}, \text{Enc}, \text{Dec})$, is given with definitions for the following functions:

- **Key Generation:** $k \leftarrow \text{KGen}(\lambda)$; which produces key k given security parameter λ .
- **Encryption:** $c \leftarrow \text{Enc}_k(m)$; encrypting plaintext message m into ciphertext c .
- **Decryption:** $m \leftarrow \text{Dec}_k(c)$; decrypting ciphertext c into plaintext message m .

Public Key Encryption

In a public key, or asymmetric, encryption scheme a different key is used for encryption and decryption. This allows a public encryption key to be widely distributed and the corresponding decryption key to be kept private. For $\mathcal{E} = (\text{KGen}, \text{Enc}, \text{Dec})$, a public key encryption scheme has the following function signatures:

- **Key Generation:** $(pk, sk) \leftarrow \text{KGen}(\lambda)$; produces public and private key-pair (pk, sk) given security parameter λ .
- **Encryption:** $c \leftarrow \text{Enc}_{pk}(m)$; encrypting plaintext message m into ciphertext c , given public key pk .
- **Decryption:** $m \leftarrow \text{Dec}_{sk}(c)$; decrypting ciphertext c into plaintext message m given secret key sk .

INDCPAGame ₁ (λ)	
1 :	$k \leftarrow \text{KGen}(\lambda)$
2 :	$b \leftarrow \$ \{0, 1\}$
3 :	$m_0, m_1 \leftarrow \mathcal{A}^{\text{OEnc}_k}()$
4 :	$c \leftarrow \text{Enc}_k(m_b)$
5 :	$b' \leftarrow \mathcal{A}^{\text{OEnc}_k}(m_0, m_1, c)$
6 :	if $b = b'$ then return 1
7 :	else return 0

Figure 2.1: IND-CPA security game. (1) A key, k is generated, as well as random bit b . (2) The adversary generates two messages. (3) m_b is encrypted. The adversary $\mathcal{A}$ wins if they can correctly identify b given the encryption of m_b . The adversary is also given access to an encryption oracle OEnc_k .

Public key schemes tend to be more resource intensive than their private key counterparts, both in terms of encryption and decryption time and key size. As such, one of their primary use-cases is in the distribution of session-specific symmetric keys. In this way people can use fast private key encryption for the bulk of their communications by first using a slower public key scheme to negotiate a shared key.

Indistinguishability under Chosen Plaintext Attack (IND-CPA)

There are many ways by which a scheme can be judged to be secure, but one of the most common is through the standard notion of Indistinguishability under Chosen Plaintext Attack (IND-CPA) security. It is a universally accepted marker of the soundness of an encryption scheme in the face of an adversary. A scheme is IND-CPA secure if it is indistinguishable under a chosen plaintext attack [21].

Breaking the term IND-CPA into its constituent parts. The scheme is indistinguishable if the adversary is unable to distinguish the encryption of two messages of their choice. When mounting a chosen plaintext attack an adversary has access to an encryption oracle for the scheme they are trying to break. They are able to interact with this oracle to encrypt any messages they like.

A common technique for reasoning about the security of a cryptographic scheme more concretely is to define a *security game*, and provide well reasoned bounds for probability that the adversary is able to win this game. In this case, a game can be constructed in which some adversary, $\mathcal{A}$, is provided with an encryption of a message randomly sampled from a set of two messages of their choice, and must determine which message was encrypted. In the CPA version of this game the adversary has access to an oracle which provides encryptions of messages of their choice. More concretely, consider the security game detailed in Figure 2.1 and the following definition.

Definition 2.4.1 (IND-CPA Secure Encryption Scheme). Consider encryption scheme $\mathcal{E} = (\text{KGen}, \text{Enc}, \text{Dec})$. The advantage of an adversary $\mathcal{A}$ is the IND-CPA security game, detailed in Figure 2.1, is defined as:

$$\text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(\lambda) = 2 \Pr[\text{INDCPAGame}(\lambda) = 1] - 1 \quad (2.7)$$

A cryptosystem $\mathcal{E}$ is IND-CPA secure if for any probabilistic polynomial time adversary $\mathcal{A}$:

$$\text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(\lambda) \leq \text{negl}(\lambda) \quad (2.8)$$

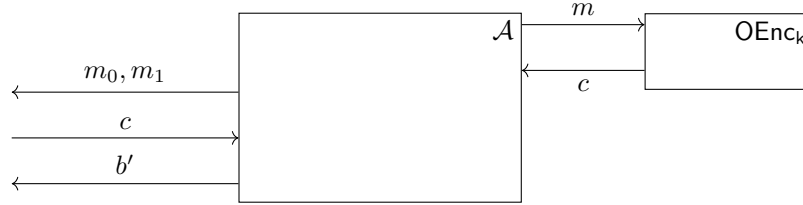


Figure 2.2: Hypothetical adversary $\mathcal{A}$, capable of winning the IND-CPA security game as detailed in Figure 2.1.

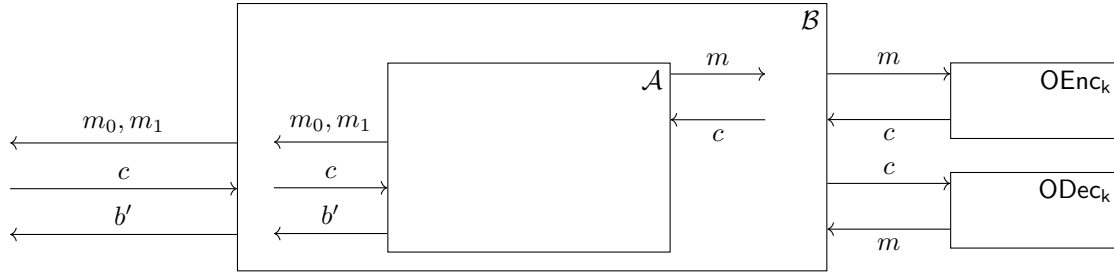


Figure 2.3: Adversary $\mathcal{B}$; capable of winning the IND-CCA security game given the existence of adversary $\mathcal{A}$.

2.4.2 Proofs by Reduction

This work features the liberal uses of a standard cryptographic proof of security known as a proof by reduction. This is a commonly used mechanism for reasoning about the security of a cryptographic construction. In a proof of this nature one considers a hypothetical adversary $\mathcal{A}$ capable of breaking some well defined property of a cryptographic construction in polynomial time with better than $\text{negl}(\lambda)$ probability. This adversary is then used as a black box to construct a new probabilistic polynomial time adversary, $\mathcal{B}$, against some computational problem widely considered to be computationally intractable.

By constructing these style of proofs it becomes easy to reason about the relative difficulty of cryptographic problems. The hypothetical reduction given above from $\mathcal{A}$ against property A to $\mathcal{B}$ against property B , shows that B is no harder than A , as given $\mathcal{A}$ it is trivial to construct $\mathcal{B}$. This is equivalent to saying that if B is considered computationally intractable, A must also be computationally intractable as B is no harder than A .

By way of demonstration consider the following proof sketches relating IND-CPA security to some related security notions. Beyond IND-CPA security, as has already been described, another commonly cited security property is indistinguishability under chosen ciphertext attack, or Indistinguishability under Chosen Ciphertext Attack (IND-CCA), security. In this setting the adversary is given an additional oracle which can provide decryptions of messages upon request, with the caveat that the challenge ciphertext cannot be sent to this oracle. Intuitively, this is a stronger notion of security than IND-CPA, as the adversary is more powerful. If an encryption scheme is IND-CCA secure it must also be IND-CPA secure, as reducing the capabilities of the adversary cannot possibly increase their ability to mount an attack on the scheme. This relationship can be demonstrated by considering a successful adversary in the IND-CPA game. This adversary can be trivially reused to win the IND-CCA game; it simply does not use the decryption oracle. This is a sketch of a proof by reduction.

More concretely, here is the proof showing that a cryptographic construction that is IND-CCA secure

must also be IND-CPA secure. Consider $\mathcal{A}$ in Figure 2.2; this describes a black-box adversary capable of winning the IND-CPA game for some scheme. $\mathcal{A}$ can be used to trivially construct $\mathcal{B}$ in Figure 2.3, capable of winning the corresponding IND-CCA game. It follows that the IND-CPA game can be no harder than the IND-CCA game. Therefore, if a scheme is IND-CCA secure, it must also be IND-CCA secure. Given formally,

$$\text{IND-CCA} \implies \text{IND-CPA} \quad (2.9)$$

or in terms of adversarial advantage,

$$\text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(\lambda) \leq \text{Adv}_{\mathcal{B}}^{\text{ind-cca}}(\lambda) \quad (2.10)$$

the advantage of an adversary in the IND-CPA game can be no more than the advantage of an adversary in the IND-CCA game, seeing as IND-CCA is no harder than IND-CPA. Proofs by reduction to known hard problems are the primary mechanism used throughout this work to reason about the ability of adversaries against the newly defined constructions presented in this work.

Another security property related to both the IND-CPA and IND-CCA properties is one that does away with indistinguishability. The setup for the games is identical to their indistinguishability counterparts, except the challenge for the adversary is not to distinguish between encryptions of two messages of their choosing, but instead to provide the decryption of a challenge ciphertext. These notions are weaker than the corresponding indistinguishability games, as one could easily see how an adversary capable of providing a ciphertext decryption could easily decrypt and therefore distinguish between encryptions of messages. This is another example of a proof by reduction.

2.5 Conclusion

This chapter has provided the common background for the chapters to follow. The included material covered important context and common vernacular for distributed networks, and specifically distributed computation networks, arbitrary precision arithmetic acceleration via RNSs, and cryptography. The security proof frameworks involving security games and adversarial advantage are of particular importance as this style of logical reasoning is used throughout this thesis. The following core content chapters each contain their own background, but in most cases these are self contained and only relate to the content presented in that chapter and can be understood in relative isolation.

Chapter 3

Proofless Verifiable Computation from Integer Factoring

In Chapter 1 a number of broadly defined attacks against a distributed computation network were identified. One such vulnerability involved the possibility of a malicious node falsifying or otherwise tampering with the output of delegated computation for which they were responsible. Adversarial activity of this sort may be deployed to force an incorrect result of the malicious node's choosing, to disrupt the computation of the issuing node, or to take credit for executing a computational task without expending any resources. This chapter presents new work designed to harden a distributed computation network against these styles of attack.

VC schemes are a class of cryptographic construction used to verify the results of a computation executed remotely [114]. In these schemes a powerful but untrusted remote node, called the Prover, executes a program on behalf of a computationally restricted node, the Verifier. The capability of a Prover to convince the Verifier of an incorrect result should reduce to the capability to compute the answer to a known computationally intractable problem.

There exists a rich body of work on VC schemes, those of most relevance to distributed computation are highlighted in this chapter. Existing schemes have some properties which make them ill-suited for distributed computation. First and foremost of these is the large additional overheads for all parties involved. Despite healthy asymptotic complexities, the use of expensive cryptographic primitives causes a dramatic multiplicative increase in the expected load of nodes when performing a VC-backed distributed computation. The most damning evidence of the impracticality of existing work is the fact that they are rarely, if ever, deployed.

After highlighting the weaknesses of existing VC schemes, this chapter then explores the VC-like properties of RNSs. Two new iterative schemes are then proposed, the first builds an intuitive idea of security, the latter includes simplifications informed by security proofs and provides a strict improvement over the former. The latter of the two schemes is analysed in-depth, highlighting the strengths and weaknesses relative to existing work. The improvement over the related work is such that the second scheme presented in this work is a viable contender for verifying the results of outsourced arithmetic over large integer rings in a distributed computation setting. The work in this chapter is summarised in [48].

3.1 Introduction

VC schemes have uses that reach far beyond distributed computation. One such application is in the domain of cloud computing, in which a less powerful client, the Verifier, delegates work to a more powerful server, the Prover. VC guarantees that the correctness of results provided by the Prover can be efficiently verified by the Verifier. The ability to efficiently validate the output of outsourced computation is a fundamental problem in securing untrusted client/server computation, but at present VC methods are not widely used in this setting. Without a viable VC mechanism to check the output of an outsourced computation, one of the few ways to build faith in the validity of a result is through the weaker notions of security, such as brand reputation and contract law [76].

The underlying insights that lead to the work presented in this chapter originated when considering if the relationship between RNS representations and integer factoring could be used to reduce the falsification of an output of a computation to the known hard problem of integer factoring. A lot of the key components are already there; a large composite number N , and small independent computational fragments which are dependent on each other to produce the correct results.

It has been discovered in this work that RNSs can be used to construct viable VC schemes with little modification. As a sketch of the protocols to follow in the remainder of the chapter, the Verifier asks the Prover to calculate the result of a computation $\text{mod } N$. Then the Verifier computes the same result $\text{mod } p$ where p is a factor of N . The two results should be congruent when reduced $\text{mod } p$. The capability of the remote worker node to cheat reduces to their ability to factor the large composite number N , which for suitably chosen N is a well established complexity assumption.

The final scheme presented in this chapter has three interesting properties over existing work:

1. Prooflessness.
2. Optimum Prover complexity.
3. The computational burden on the Verifier is never worse than evaluating the target computation locally, plus a single modular reduction and integer comparison.

The first and most novel property is that the scheme is *Proofless*. Unlike in the related work, the raw result of a computation is the only output required from the Prover. Other schemes require a proof to be produced in tandem to the computational output, which is then checked. Sometimes the proof provided by the related work is not strictly separate, but an encoded form of the computational output that requires the Verifier to extract the computational result, providing an error if the result is incorrect. This is another form of additional proof, as the encodings must be larger than the output of the raw target computation. The additional space required for proofs in related work is given in a later chapter discussing the properties of related work in more detail, and can be seen in Table 3.1.

Secondly, the scheme has the best possible Prover complexity, as they do no more work than if they were evaluating an unverified delegated computation. When considering distributed computation there are special considerations to be made that are often omitted in the related work. Specifically, in reality it cannot be assumed that the untrusted remote server is computationally unbounded. Unlike the clear delineation between the weaker client and more powerful server in the client/server architecture, in distributed computation nodes have a less clearly delineated variety of compute power. Furthermore, some or all nodes may spend time as both the work issuing Verifier, or for different tasks as the work completing Prover. Clearly, additional computational burden of a VC scheme on both the Verifier and Prover should be carefully considered in the

distributed computation setting. Even in the standard client/server setting the idea of a computationally unbounded untrusted server is only loosely true. It is often the case that the Verifier is paying in some way for the Prover's time and as such the Verifier is motivated to reduce their burden on the Prover as much as possible. This is one of the failings in existing VC work; as they often assume the Prover is arbitrarily powerful and care little about how much extra work they have to do to compute a result and also produce a proof of correctness. At best the related work accepts a multiplicative increase in the computational burden on the Prover. In the new scheme presented in this work, the computational burden on the Prover is unchanged as compared to unverified computation, which is in stark contrast to existing VC schemes. This has two implications. First, usually there is some incentive to reduce the work on remote worker nodes, such as paying for their time, or energy expenditure. In this respect the new work presented here clearly arrives at an optimum solution, seeing as their workload is unchanged from unverified outsourced computation. The second is an interesting property; that the remote worker node is blind to the fact they are participating in a VC scheme; as their input, workload, and output is identical to an unverified remote computation protocol.

Thirdly, although the new scheme has worse asymptotic properties than the related work, the computational burden on the work-organising node with the new VC scheme is never any worse than if they simply re-executed the program themselves to check the result. This is not always a given in the related work. For specific programs, such as those only ever executed once, the related work is more expensive than naively re-executing a program. An expensive per-program setup phase requires repeat executions for these VC schemes to achieve a degree of practicality. By not requiring a setup phase at all these expensive steps are avoided, at the cost of a verification step with worse asymptotic properties.

3.1.1 Contributions

Two VC schemes are presented in this chapter, the former is an intermediate scheme with the later offering a strict improvement over the former. As such, omitting the intermediate scheme, the contributions of this chapter can be summarised as:

- The specification of a novel VC scheme which to the best of my knowledge is the first VC scheme to have the following three properties:
 - Is proofless, the raw computational output of the target program is verified without any additional data from the Prover.
 - Is Prover optimal.
 - Is built on the well studied problem of integer factoring.
 - The computational complexity for the Verifier is never worse than executing the target computational locally, plus a single modular reduction and integer comparison.
- Proofs that the scheme presented satisfies the standard VC properties of *Completeness*, and *Soundness*. As well as a proof that the scheme is *Outsourceable* for target programs consisting of arithmetic over large integers.
- The scheme has extremely low latency, allowing much of the verification work to be done asynchronously, with only a single modular reduction required by the Verifier upon receiving the results from the Prover.
- The scheme is relatively easy to implement, only requiring some minor extra machinery for the Prover over an existing outsourced computation protocol.

For large composite number $N = \prod P_N$ and target computation encoded as an arithmetic circuit C_N , the VC scheme presented in this chapter has the following properties:

- A small one-time setup, **KGen**, independent of the size of the target computation, $\mathcal{O}(|P_N|)$, consisting of the generation of a suitable instantiation of the integer factoring problem.
- A zero-cost per-program pre-processing step, **ProbGen**.
- A verification step, **Vf**, with computational complexity which is a constant factor of the computational complexity of the target program, $\mathcal{O}(|C_p|)$.
- No additional Prover computational burden as compared with unverified computation.
- No additional bandwidth as compared with unverified computation.
- Adding VC functionality to an existing unverified remote computation framework only requires implementing the **KGen** and **Vf** functions, as the other required standard VC functions are defined as an identity relation on their inputs.

The downside of the schemes presented in this chapter in relation to the related work are worse asymptotic properties for the **Vf** function. For programs executed many times, these schemes are almost certainly not the most suitable approach. However, where related work requires repeat executions of the same program to be considered *Outsourceable*, this work is never worse than the naive duplication of effort.

3.1.2 Threat Model

This work is focused on a threat model in which polynomially bounded adversaries attempting to falsify the output of a delegated computation. Some of the related work focuses on weaker threat models in which the adversary is bounded by the computational complexity of the delegated program, a model that simulates adversarial behaviour where Provers are trying to get away with doing less work than what is delegated to them. Here, however, the VC scheme is designed to be sound against much more powerful adversaries, willing to invest serious computational effort in forging the results of delegated computations. These adversaries, as is convention, are bounded polynomially in the security parameter λ .

3.1.3 Assumptions

This work makes the standard and well studied assumption of the intractability of integer factoring [80].

Assumption 1 (Intractability of Integer Factoring). Producing any non-trivial factors of large composite integer N is computationally intractable.

3.2 Background

First, this background establishes the standard VC nomenclature. Then the mechanics of arithmetic circuits are detailed as, mirroring other works in the field, computations are represented as arithmetic circuits. This is followed by required background on the integer factoring problem, which constitutes the foundational complexity assumption for this cryptographic construction. The scheme ultimately presented in this chapter is a new approach to VC that is built from connecting an RNS-style representation of operands in a computation to the integer factoring problem.

3.2.1 Notation

Along with the notation presented in Chapter 2, this subsection details notation specific to this chapter. As is standard in VC work, the local client that has issued the work is referred to as the Verifier, and the remote server performing the outsourced execution is known as the Prover. B and C are reserved for binary circuits and arithmetic circuits respectively. In this work arithmetic circuits are used, but binary circuits are referenced when discussing related work. When an arithmetic circuit C operates over a ring $\mathbb{Z}_x$, such that each arithmetic operation is resolved mod x , the notation C_x is used.

3.2.2 Verifiable Computation

A comprehensive formal definition of the VC problem can be found as part of [58], the following function definitions are adapted from this work, with some choice simplifications which are described after. Given security parameter λ , a VC scheme, $\mathcal{V} = (\text{KGen}, \text{ProbGen}, \text{Compute}, \text{Vf})$, is defined by the following functions:

- **Key Generation:** $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(\lambda)$; generate a public and secret key pair.
- **Problem Generation:** $\sigma \leftarrow \text{ProbGen}_{\text{pk}}(f, x)$; encode arbitrary function f and input x into problem σ .
- **Compute:** $r \leftarrow \text{Compute}(\sigma)$; process problem σ , producing a result r .
- **Verification:** $\top$ or $\perp \leftarrow \text{Vf}_{\text{sk}}(\sigma, r)$; verify the result r , given problem σ . $\top$ is an accepted verification, $\perp$ indicates a rejection.

The function definitions are modified slightly from those presented in [58], as the new constructions in this chapter have comparatively simple function definitions. In the related work **Compute** often requires access to the public key pk , whereas the versions of this function presented in this work do not. Similarly, in related work, **Vf** is required to output the correct result of the computation or $\perp$. Instead, in this work it is the case that r is the result of the computation with no further processing required, so **Vf** need only accept ($\top$) or reject ($\perp$) the result. One of the largest divergences from related work is that the key generation function does not need to know the target computation, beyond some loose bounds on the operand size. This is in contrast to much of the related work, in which key generation is bespoke for each target computation f . These simplifications causes subtle changes in the other function signatures, and in the security proofs that follow later in this chapter.

These functions above are orchestrated into a protocol between a Verifier and a Prover. Figure 3.1 provides an overview of this exchange. Key generation, problem generation, and verification are all executed by the Verifier, and compute is executed by the Prover.

A working VC scheme must accept the correct results of a computation, and reject any incorrect results. These properties are called *Completeness* and *Soundness* respectively. *Completeness* is a measure of whether a VC scheme behaves correctly when the Prover is acting honestly.

Definition 3.2.1 (Completeness). For function f , input x , encoding $\sigma \leftarrow \text{ProbGen}(f, x)$, and honestly generated result $r \leftarrow \text{Compute}(\sigma)$

$$\Pr[\text{Vf}_{\text{sk}}(\sigma, r) = \top] = 1 \quad (3.1)$$

The *Soundness* property relates to the security of a VC scheme in the face of an adversarial Prover. Consider the security game outlined in Figure 3.2. In this game an adversary, $\mathcal{A}$, is provided with a public

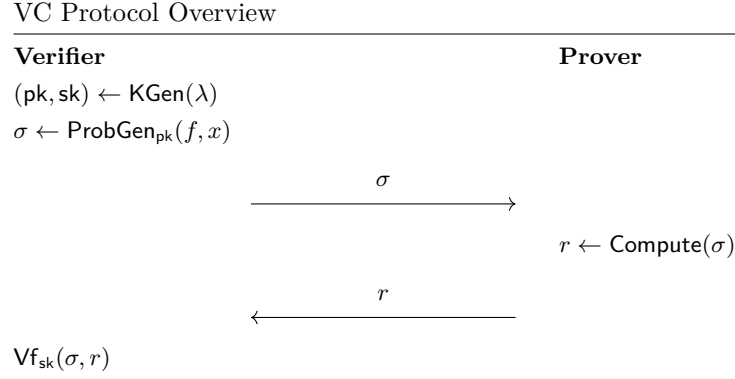


Figure 3.1: VC protocol overview. A public and private key pair is generated, a target function f and input x are encoded as problem σ , and this is sent the Prover. The Prover then evaluates the problem, generating output r , which is sent to, and verified by, the Verifier.

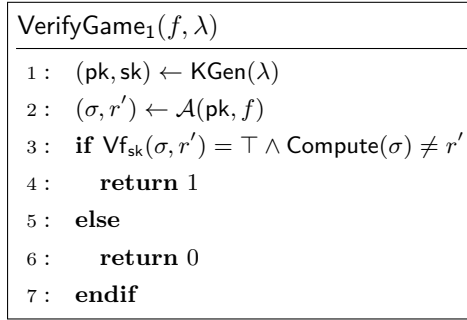


Figure 3.2: Verify security game for VC schemes. (1) Generate key pair, (2) send f and pk to adversary, which returns an encoded problem σ and forgery r' , (3) the adversary wins if r' is not the correct result of $\text{Compute}(\sigma)$ but is accepted by the Verifier regardless.

key pk and function f , and wins the game if they are able to construct a valid encoded problem σ and an incorrect forged output r' , such that Vf incorrectly returns $\top$. Throughout this work the term *forged result* or *forgery* is used to refer to an incorrect result provided by a malicious Prover. A VC scheme has the *Soundness* property if the probability of $\mathcal{A}$ winning the VerifyGame is negligible.

The VerifyGame allows the adversary to select σ , which seemingly misaligns with the construction of a VC scheme as presented in this chapter - in that σ is generated by the Verifier. However, this is a classic example of empowering an adversary, such that in subsequent proofs a system is proven secure even under this empowered adversary. Specifically in this case, if the advantage of an adversary in the VerifyGame is negligible when the adversary is able to select σ , then “in the wild” when σ is generated by the Verifier, an adversary will be able to do no better than their more powerful counterpart. A similar relationship between empowered adversaries can be seen in the definitions of IND-CPA and IND-CCA security as presented in Section 2.4.

Definition 3.2.2 (Soundness). For a VC scheme, the advantage of an adversary to VerifyGame , as detailed in Figure 3.2, is defined as:

$$\text{Adv}_{\mathcal{A}}^{\text{verify}}(f, \lambda) = \Pr[\text{VerifyGame}(f, \lambda) = 1] \quad (3.2)$$

KGen 1 : return (null,null)	ProbGen (f, x) 1 : $\sigma \leftarrow (f, x)$ 2 : return σ	Compute (σ) 1 : $(f, x) \leftarrow \sigma$ 2 : $r \leftarrow f(x)$ 3 : return r	Vf (σ, r) 1 : $r' \leftarrow \text{Compute}(\sigma)$ 2 : if $r = r'$ then 3 : return $\top$ 4 : else 5 : return $\perp$ 6 : endif
--	---	---	---

Figure 3.3: KGen, ProbGen, Compute, and Vf for a trivial but non-*Outsourceable* VC scheme.

A VC scheme is secure for a function f , if for any probabilistic polynomial time adversary $\mathcal{A}$

$$\text{Adv}_{\mathcal{A}}^{\text{verify}}(f, \lambda) \leq \text{negl}(\lambda) \quad (3.3)$$

By way of demonstration, a trivial VC scheme can be constructed which has both the *Completeness* and *Soundness* properties. In this trivial scheme the Verifier duplicates the work executed by the Prover to check the results. Figure 3.3 outlines such a scheme, in which **Compute** is called during the execution of **Vf**.

The duplication of effort in the trivial scheme is clearly suboptimal, and there is no discernible benefit to outsourcing the computation in the first place. The challenge therefore, is to create a scheme in which the total computational burden on a Verifier is less than if they were to independently compute $f(x)$. A scheme with this property is *Outsourceable*.

Definition 3.2.3 (Outsourceable). For any function f , the total time required to compute KGen, ProbGen and Vf is less than the time required to compute $f(x)$ for any x .

For one shot applications the cost of ProbGen and Vf must be less than the time taken to evaluate f . For functions that are expected to be executed more than once, the combined cost must be asymptotically smaller than repeatedly evaluating f . Related work usually only satisfies this looser bound.

Execution Cost

In this chapter the execution costs of the protocol is discussed from both the perspective of the Prover and also the perspective of the Verifier.

Definition 3.2.4 (Prover Computational Cost). For an arbitrary function f and input x , the *Verifier cost* is the combined computational costs of evaluating KGen, ProbGen, and Vf.

Definition 3.2.5 (Verifier Computational Cost). For an arbitrary function f and input x , the *Prover cost* is the cost to evaluate Compute.

3.2.3 Arithmetic Circuits

As is common in VC work, a computation to be verified is expressed as an arithmetic circuit, C . An arithmetic circuit is an acyclic directed graph of arithmetic operations. Nodes in this graph represent arithmetic gates, which perform some arithmetic operation on their input data. The input and output of these gates are often called wires. Figure 3.4 is a toy example of an arithmetic circuit for a simple program. For a given set of inputs the circuit is evaluated by iteratively processing the input wires to a gate based on the gate's

designated operation. One or more output wires are labelled as the output to the entire circuit. A single output wire can fan into many input wires.

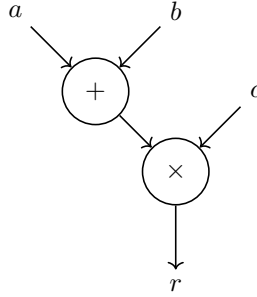


Figure 3.4: An arithmetic circuit representation of $r = c(a + b)$.

A related computational model often seen in other VC work is the binary circuit, B . These function similarly to arithmetic circuits, except the gates are binary operations, such as **AND** or **OR** and the input and output of these gates is restricted to $\{0, 1\}$. As compared with arithmetic circuits, binary circuits can express comparison operations in fewer gates, but otherwise arithmetic circuits are more concise than binary circuits, as for many programs the number of wires required to be expressed as an arithmetic circuit is approximately four orders of magnitude smaller than the same program represented as a binary circuit [103]. It is widely accepted that arithmetic circuits, and their binary circuit cousins, are a relatively good model for general computation [110].

3.2.4 Integer Factoring Assumption

The security of this scheme reduces to the intractability of the factoring integers. The difficulty of integer factoring is a widely accepted standard cryptographic assumption [80]. Given Assumption 1, producing any non-trivial factors of the large composite integer N is computationally intractable. While not **NP-Complete**, current estimates class integer factoring as an **NP-Intermediate** problem. There is no polynomial time algorithm for factoring numbers and the current state-of-the-art uses a General Number Field Sieve (GNFS) to achieve subexponential time [31].

Unfortunately integer factoring is not post-quantum secure, as running Shor’s algorithm on a quantum computer can factor integers in polynomial time [104]. Current practical limitations on quantum computing [8] indicate a great deal of longevity in cryptosystems built on the difficulty of the integer factoring problem.

The formulation of an instance of the integer factoring problem is based on the current best known algorithms to factor N . To better reason about the security of this scheme the generation of a suitably difficult factoring problem has been abstracted to an oracle,

$$(N, P_N) \leftarrow \text{PrimeFactorProblem}(\lambda) \text{ s.t. } N = \prod_{p \in P_N} p \quad (3.4)$$

which produces a suitably difficult instance of the integer factoring problem to satisfy security parameter λ . The security game detailed in Figure 3.5 is used in later security proofs. The advantage of an adversary in this game is defined as follows:

FactorGame ₁ (λ)	
1 :	$(N, P_N) \leftarrow \text{PrimeFactorProblem}(\lambda)$
2 :	$a \leftarrow \mathcal{A}(N)$
3 :	if $N/a = b$ for integers $b, a \notin \{1, N\}$
4 :	return 1
5 :	else
6 :	return 0
7 :	endif

Figure 3.5: Factoring security game for the integer factoring problem. (1) Generate suitably hard factorisation problem, (2) send N to the adversary and receive result a , (3) the adversary wins if a is a non-trivial factor of N .

Definition 3.2.6 (FactorGame Advantage). For the integer factoring problem the advantage of an adversary to FactorGame, detailed in Figure 3.5, is defined as:

$$\text{Adv}_{\mathcal{A}}^{\text{factor}}(\lambda) = \Pr[\text{FactorGame}(\lambda) = 1] \quad (3.5)$$

3.3 Related Work

A great deal of work has already been done attempting to create a suitable mechanism to check the results from a remote computation. These solutions range from concrete number theoretic constructions [11, 14, 39, 58, 63, 95, 100, 103] through to economic solutions wherein the cost of sending incorrect results is set discouragingly high [19, 65]. A wide selection of existing VC schemes are presented in this section and constitute the various state-of-the-arts of the field. Economic solutions are omitted from the discussion as they suggest a vastly different context, requiring economic machinery to levy fines and compensation, while only providing a minimal degree of security. A sufficiently motivated adversary would have no problems paying levied fines to corrupt computations of other parties. The remaining categories explored here are:

1. Audit-based solutions.
2. Trusted Platform Modules (TPMs).
3. Number theoretic solutions.

The first family of approaches is that of audit solutions, where tools are used to gain insights into the execution flow of a target program. The next is that of TPMs, where golden hardware and software states can be verified and the correctness of results implied by induction. Finally the various number theoretic VC constructions are described, in which the ability to cheat is reduced to solving computationally intractable problems. This final class of solution includes works most closely related to the protocol presented in this chapter, however none of these existing works are constructed from the same foundational assumption used here, and often they require less well-studied assumptions. For more information on varied available verifiable computation constructions the interested reader is directed to [114].

3.3.1 Auditing

The first concrete anti-cheat mechanisms were built around the idea of performing an audit on the work process. During execution, regular stack traces can be saved and returned alongside the results. Random pairs of consecutive stack traces can then be validated by executing the program between these two checkpoints and validating the later stack trace as correct given the former [88]. Faith is built in the correctness of the answer by checking enough of the pairs provided by the Prover.

An interesting idea for detecting if a program was run at all involves introducing random interactive processes at compile time [112]. In this work automated randomised interactive elements are added to a program during compilation, such that at points known to the Verifier during the program execution the Prover will interact with the Verifier in a known way. Such a tool is not designed to defend against a powerful malicious adversary capable of running the program and sending falsified results regardless. Such an adversary may have motives beyond circumventing the cost of executing the program.

3.3.2 Trusted Platform Modules

A pragmatic solution to verifying the execution of a program is to produce a golden hardware and software state and then create a mechanism for a remote computer to prove that they are in the golden state when performing a computation. TPMs have allowed this style of validation to be possible, with attestation procedures that can provide witness to the entire hardware/software stack [7]. Although this is a viable solution, it does not provide much flexibility in implementation. To build a solution where remote parties are attesting with TPMs requires a sufficiently benevolent and trusted device vendor willing to build these modules on the behalf of participants, or someone with enough resources at their disposal to do it themselves. The creation of a golden state requires trusting every party in the hardware and software development life cycle.

3.3.3 Number Theoretic Solutions

A more rigorous approach to VC involves formulating programs in such a way that the capability to cheat without detection is reduced to known hard problems.

Program Checking

Early number theoretic solutions concerned themselves with automated checking of results given some possibility of non-malicious errors. The field of *Program Checkers* builds this functionality for a suite of standard problems. For specifically chosen problems, such as numerical functions [25], or sorting and greatest common divisor [24], the authors construct bespoke checking programs that certify a program was executed correctly without bugs. Program checkers are designed to detect accidental errors in a computation and so are not secure against a malicious adversary willing to invest computational effort to create forgeries.

Efficient but Problem Specific Solutions

Some problems have specific VC constructions which can provide a high guarantee of correctness, with a high degree of security and approachable resource requirements for both the Verifier and the Prover. Unfortunately, these constructions are bespoke and only work for a specific problem, or subset of problems.

An example of a problem where this approach is suitable includes computation across polynomials [68], ideal for health-related computation and a myriad of other simple classification problems. Verifiable computation across polynomials can be achieved by pairing the somewhat-homomorphic BGN encryption scheme [28] with a publicly verifiable computation scheme across polynomials [94]. Similarly, constructions also exist for verifying searches over encrypted data [108], modular exponentiation operations [20], polynomial function evaluation [27], and matrix multiplication [12].

Probabilistically Checkable Proofs (PCPs)

PCPs are a general class of interactive proof systems wherein a non-deterministic Verifier is able to verify a proof in polylogarithmic time in the size of the proof. By definition they seem as if they are ideally suited to the VC problem, and in fact early attempts at a system for verifying general computation involved using known PCPs [14, 63]. Unfortunately proofs in this setting tend to be very long, and so verification in polylogarithmic time can often lead to proofs that take longer to verify than naively re-executing the target program. There has been some work modifying these schemes to be more efficient by tailoring them to specific given problems [44], but even with more modern cryptographic tools the computational break-even point is rarely reached. One modern example uses argument systems, a variant of PCPs in which the Prover answers queries from the Verifier interactively [103]. Modern PCPs have a great deal of implementation complexity, which deter adoption and makes a bug-free implementation difficult.

Another of the first fully formed general PCP VC schemes involved combining Yao’s Garbled Circuit construction [113] with a fully homomorphic encryption scheme to allow reuse of the previously-single-use VC properties of the garbled circuits [39, 58]. By decoupling the setup from the input data, the expensive setup can be amortised across multiple executions of the program with different input data.

Multi-Party Computation (MPC) is a closely related field of inquiry to VC, in which two groups work together to securely compute a function on a shared input without revealing the input data to the other party. A natural question is: Is it possible to construct VC from MPC? The answer is yes, the construction involves using an MPC scheme to compute the signature of the output of an outsourced function [11].

Some strict PCP-based VC schemes, such as Pinocchio [95], are close to practical for a reduced domain of possible programs, albeit with some non-standard assumptions. Verification is cheap, but the setup costs are very close to the computation cost of the target program. As such program encodings must be reused with new inputs to reduce the average computational burden on the Verifier per execution.

Interactive Oracle Proofs (IOPs) are a generalisation of PCPs, that require a Prover to allow interactive querying from the Verifier. These interactive proof systems have been shown to allow for much more efficient program verification of boolean circuits than their non-interactive counterparts in the real world [100]. In this case, more succinct arguments are created by the removal of a requirement to expensively encode the entire computation.

Succinct Non-Interactive Proofs

Succinct non-interactive proofs aim to improve the asymptotic properties of existing Zero Knowledge Proof (ZKP) schemes to provide the performance properties required by a VC scheme. One of the most widely acclaimed of these is Bulletproofs [34]. Bulletproofs have good asymptotic properties, with verification linear in the circuit size, but expensive cryptographic operations mean that in practice they are rarely viable, often taking orders of magnitude longer than vanilla computation. Some more recent work, Libra [111], has improved practical performance requiring $\mathcal{O}(d \log |C|)$ time to verify for circuits C of depth d , but only

produces succinct arguments for some specifically structured arithmetic circuits. Interestingly Libra is “Prover optimal” with Prover time of $\mathcal{O}(|C|)$, this property contrasts with the Prover optimally property of the work in this chapter in which the Prover is required to do *exactly* $|C|$ operations.

3.3.4 Identified Problems

At best, the existing solutions front-load the additional computational complexity to a setup phase, and then have very efficient verification. This allows for amortised complexity over multiple executions of the program, with different input data.

As shown in Table 3.1, schemes tend to add some large computational overhead for the Prover. All existing schemes add, at a minimum, a multiplicative factor to the Prover workload over vanilla computation. All related work, save Pinocchio [95], have very large proofs which scale with the circuit size. There is a clear gap to be explored with a VC scheme that is extremely cheap to setup with tiny proofs, possibly at the cost of being more expensive to verify. Likewise, the current best “Prover optimal” solution requires the Prover to expend effort linear in the size of the arithmetic circuit, development of a scheme with even less Prover overhead would have immediate benefits for Provers constrained in any way. All related works in Table 3.1, except for Bulletproofs rely on non-standard cryptographic assumptions. In practice these assumptions seem reasonable, and are relatively well studied, but they are not taken to be as sound as standard cryptographic assumptions such as the Discrete Logarithm Problem (DLP) or the hardness of integer factoring.

Despite the rich body of existing work on VC, the most telling evidence that more work is required is that modern volunteer distributed computation deployments, such as Folding@home [18], use naive work duplication to validate outsourced computation.

3.4 Intermediate Verifiable Computation Scheme

Two related VC constructions are presented in this work, where the first is an intermediate step to the second. The intuition behind this intermediate VC construction is to combine two techniques in a new way. These techniques are RNSs and existing audit-based solutions. When using this VC scheme, given the output to some arithmetic circuit evaluated in a $\mathbb{Z}_N$ on some input x , $C_N(x)$, the Verifier can independently execute different partial fragments of the RNS representation of the same circuit $C_p(x)$ for $p \in P_N$. The Verifier can progressively execute more fragments until they are confident in the validity of the result provided by the Prover. By virtue of the fact RNSs are used to accelerate integer arithmetic, computing via the RNS representation will take no longer than computing without it. Because the Verifier only executes a subset of the total partial computations this scheme should be *Outsourceable*.

After introducing the VC scheme definition, a proof is then provided for the construction’s *Soundness*. The initial intuition behind the *Soundness* of the scheme is grounded in the idea that the Verifier is able to build up faith in the correctness of a result by checking a sizeable proportion of the work done by the Prover. However, the proof shows that is at least as hard to produce forgeries in this scheme as it is to factor large integers, and therefore only a single element $p \in P_N$ needs checking. This leads to the complete scheme in Section 3.5.

	ProbGen	Compute	Vf	Proof Size	Complexity Assumption
Pinocchio [95]	$\mathcal{O}(C)$	$\mathcal{O}(C)$	$\mathcal{O}(\log x + \log r)$	$\mathcal{O}(1)$	q-PKE [67], q-PDK [67], q-SDH [26, 57]
[100]	$\mathcal{O}(B)$	$\mathcal{O}(B)$	$\mathcal{O}(\log x + \log B)$	$\mathcal{O}(\log B)$	Low Complexity Cryptographic Hash Functions [10]
Bulletproofs [34]	0	$\mathcal{O}(C)$	$\mathcal{O}(C)$	$\mathcal{O}(\log C)$	DLP [86]
Libra [111]	$\mathcal{O}(\log x)$	$\mathcal{O}(C)$	$\mathcal{O}(d \log C)$	$\mathcal{O}(d \log C)$	q-SBDH, eq-PKE
No VC (Insecure)	0	$\mathcal{O}(C)$	0	0	
Naive Duplication	0	$\mathcal{O}(C)$	$\mathcal{O}(C)$	0	

Table 3.1: Computational complexity of a selection of state of the art VC schemes, compared to unverified remote computation, and naive duplication. x is the input to arithmetic circuit C with depth d and result r . Schemes over boolean circuits use B instead of C .

3.4.1 Outline

Given an arithmetic circuit C_N with some input x , where N is a large composite number $N = \prod P_N$. The Prover is sent C_N and x , evaluates $C_N(x)$ and returns the result, r , to the Verifier. The Verifier then selects $Q \subset P, Q \neq \emptyset$. For all elements $q \in Q$ the Verifier independently evaluates $r_q \leftarrow C_q(x)$. If, r the result provided by the Prover is correct, then $r \equiv r_q \pmod{q}$. If for any q it is the case that $r \not\equiv r_q \pmod{q}$, r must be incorrect. By setting $|Q|$ to be fairly large, the Verifier can build up faith in the answer provided by the Prover. Despite this intuitive notion of security, it turns out the *Soundness* proof shows this is a substantial underestimate of the security of the construction.

3.4.2 Definition

The definition of this construction is given as definitions to the standard VC definitions in Figure 3.6, with a text-based description of these mechanics provided in the remainder of this subsection.

First **KGen**, in advance the Verifier preselects a computation they wish to evaluate C_N . They then randomly generate a set of coprime moduli P_N , multiplying these together to produce $N \leftarrow \prod P_N$. N must be large enough that any expected intermediate results of the computation are smaller than N .

The Verifier then executes **ProbGen**, which involves no more work than packing C_N and the input x into a tuple σ and sending this to the Prover.

$$\sigma \leftarrow (C_N, x) \tag{3.6}$$

The Prover then unpacks σ , evaluates $C_N(x)$

$$(C_N, x) \leftarrow \sigma \tag{3.7}$$

$$r \leftarrow C_N(x) \tag{3.8}$$

and sends r back to the Verifier.

In order to verify the result from the Prover, the Verifier independently calculates a subset of the residual values that would comprise r decomposed into an RNS. They do this by sampling $Q \subset P_N$, such that the probability of detecting a forgery, $\frac{|Q|}{|P_N|}$, is large enough for the Prover to be convinced of the correctness of r . Then the circuit C is evaluated under each element of Q to produce a set of results R .

$$R = \{C_q(x) \mid q \in Q\} \tag{3.9}$$

Then, r is checked to be congruent to each $r_q \in R$ under q .

$$r \equiv r_q \pmod{q}, \forall r_q \in R \tag{3.10}$$

If this holds for all $r_q \in R$ then **Vf** returns $\top$ accepting the result, otherwise it returns $\perp$ rejecting it.

The decomposition of this scheme into its constituent definitions for the standard VC functions is given in Figure 3.6. These can be composed into a protocol as previously shown in Figure 3.1.

3.4.3 Correctness and Outsourceability

The intuitive *Correctness* of this scheme and degree to which this scheme is *Outsourceable* is based on the close resemblance to an RNS. For the sake of brevity proofs for these properties are only sketched as they

KGen(λ)	ProbGen $_N(C, x)$	Compute(σ)	Vf $_{P_N}(\sigma, r)$
1 : $P_N \leftarrow \$\mathbb{Z}$	1 : $\sigma \leftarrow (C_N, x)$	1 : $(C_N, x) \leftarrow \sigma$	1 : $(C_N, x) \leftarrow \sigma$
2 : $N \leftarrow \prod P_N$	2 : return σ	2 : return $C_N(x)$	2 : $Q \leftarrow \$P_N$
3 : return (N, P_N)			3 : for $q \in Q$ then
			4 : $r_q \leftarrow C_q(x)$
			5 : if $r \not\equiv r_q \pmod p$ then
			6 : return $\perp$
			7 : endif
			8 : endfor
			9 : return $\top$

Figure 3.6: The KGen, ProbGen, Compute, and Vf function definitions for the initial VC scheme.

are identical in form to the proofs given for the improved scheme presented in Section 3.5.

The scheme must have the *Correctness* property, as with honest participants r must be the correct output of the circuit evaluation performed by the Prover. By the Chinese remainder theorem, r will always be congruent to the partial values produced by the Verifier as they perform what is in fact a partial RNS evaluation of the circuit.

To be *Outsourceable* the Verifier must always do less work than it would take to calculate the result of the arithmetic circuit C_N . As RNSs are a common acceleration technique [46], it stands to reason that the complexity of each partial evaluation of the circuit done by in RNS form must be no more than the computational complexity of calculation done not in RNS form. As such the Verifier performs a fraction of the work done by the Prover given by:

$$\frac{|Q|}{|P_N|} \leq 1 \quad (3.11)$$

However, this result only holds if the operands used approach N in size. The reduction in computational complexity stems from the reduced complexity of working with smaller operands allowed by an RNS. In the extreme case, if the operands are already very small, $\forall p \in P_N, x \ll p$, each intermediate circuit evaluation performed by the Verifier will be equivalent to evaluating the circuit modulo N . As such these scheme is not strictly *Outsourceable* as it is very possible for an arbitrary computation to require a multiplicative increase in the amount of work done by the Verifier over simply evaluating C_N themselves. This is improved in the full construction that follows in Section 3.5.

3.4.4 Soundness

The intuition behind the scheme's *Soundness* is that the Verifier independently checks a subset of the partial RNS representation values of the result r , and therefore the probability of detecting a forgery is proportional to the numbers of elements checked.

This, however, is a gross understatement of the security of the system. Consider the following theorem.

Theorem 3.4.1 (Soundness). For this VC scheme, the advantage of a computationally bounded adversary in the verification game, $\text{Adv}_{\mathcal{A}}^{\text{vf}}(\lambda)$, is at most that of the advantage of a computationally bounded adversary in the integer factoring problem, $\text{Adv}_{\mathcal{B}}^{\text{factor}}(\lambda)$, given by $N = \prod P_N$.

$$\text{Adv}_{\mathcal{A}}^{\text{vf}}(\lambda) \leq \text{Adv}_{\mathcal{B}}^{\text{factor}}(\lambda) \quad (3.12)$$

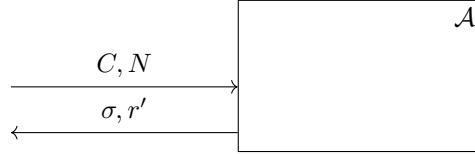


Figure 3.7: Adversary $\mathcal{A}$; capable of producing forged results r' in this VC scheme to `VerifyGame`, as defined in Figure 3.2, with advantage $\text{negl}(\lambda) < \text{Adv}_{\mathcal{A}}^{\text{vf}}(\lambda) < 1$.

Proof. Consider a hypothetical adversary against the VC scheme presented in this section, $\mathcal{A}$, given in Figure 3.7. This adversary produces forgeries $r' \neq C_N(x)$ with a probability greater than 0 of going undetected during Vf. This means that for at least one element $p \in P_N$:

$$\exists p \in P_N, r' \equiv r_p \pmod{p} \quad (3.13)$$

As r_p is independently calculated by the Verifier, $\forall p \in Q, r \equiv r_p \pmod{p}$, where $Q \subset P_N$ and r is the genuine result $r = C_N(x)$. It follows that:

$$\exists p \in P_N, r' \equiv r \pmod{p} \quad (3.14)$$

So, for at least one element $p \in P_N$, the forgery r' is equivalent to the actual result r , when reduced mod p . S is the non-empty set of values $p \in P_N$ under which $r' \equiv r \pmod{p}$:

$$S = \{p \mid p \in P_N, r \equiv r' \pmod{p}\} \quad (3.15)$$

If r is a forgery that can possibly go undetected, from Equation 3.13 there must be at least one element that satisfies the requirements of an membership in set S , and so $S \neq \emptyset$.

It follows, as S is not empty and $r \equiv r' \pmod{p_s}$ for $p_s \in S$ that for some integer a :

$$r = r' + a \prod S \quad (3.16)$$

$$|r - r'| = a \prod S \text{ for } a > 0 \quad (3.17)$$

It is then trivial to extract the non-trivial factor of N given by $\prod S$, as the greatest common divisor of $|r - r'|$ and N . If $\mathcal{A}$ can produce forgeries for this VC scheme, the above process can be used to extract non-trivial factors of N . This reduction is shown diagrammatically in Figure 3.8. This new adversary, $\mathcal{B}$, has no interest in VC and instead wishes to leverage $\mathcal{A}$ to produce non-trivial factors of N . As such it initially arbitrarily samples C from the set of possible circuits F , taking care to select one that is straight forward to evaluate, for it does not matter what C is at all.

Given this reduction, the integer factoring problem given by $N = \prod P_N$ is no more difficult than producing a forgery to this VC scheme.

$$\therefore \text{Adv}_{\mathcal{A}}^{\text{vf}}(\lambda) \leq \text{Adv}_{\mathcal{B}}^{\text{factor}}(\lambda) \quad (3.18)$$

□

The interesting implication of this proof is that the reduction holds even if $|S| = 1$. So, to produce a forgery under even a single $p \in P_N$ is computationally intractable if factoring N is computationally intractable. This proves that given N is hard to factor, not only is the security of the scheme incredibly high but also that

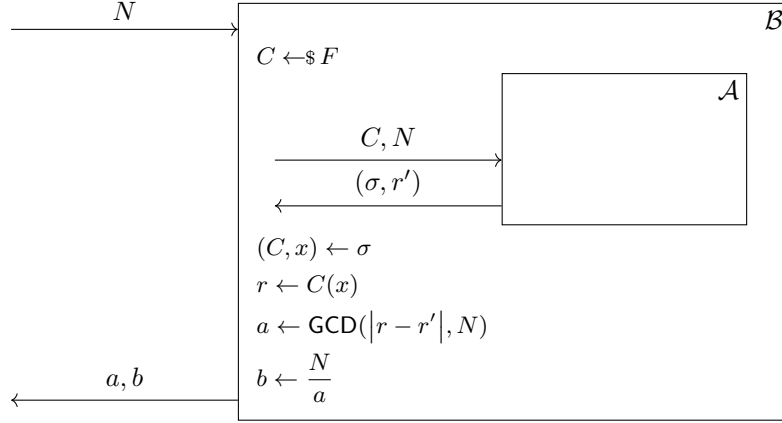


Figure 3.8: Adversary $\mathcal{B}$, capable of calculating two non-trivial factors to any composite integer given the existence of Adversary $\mathcal{A}$.

the set of checked elements of N , Q need only contain a single element. Thus, $\mathbf{Vf}$ need not check an entire subset of $Q \subset P_N$, but instead need only check a single $p \in P_N$. This simplification follows in the subsequent section.

3.5 Verifiable Computation Scheme from the Hardness of Integer Factoring

The VC scheme presented in this section improves on the previous VC scheme with a simplified $\mathbf{Vf}$ step. As indicated by *Soundness* Theorem 3.4.1, the Verifier need only check $r' \equiv r_p \pmod p$ under a single element $p \in P_N$. This should dramatically improve the computational burden on the Verifier, making it easier to argue for the system's *Outsourceability*, without harming the *Soundness* of the scheme. As this is the most advanced iteration of this scheme to-date it is followed by exhaustive proofs for VC properties: *Completeness*, *Soundness*, and after considering the computational and bandwidth complexity, a proof for the systems *Outsourceability*.

The Outline and Definition to follow share many features to the previous intermediate construction. They are described in their entirety for the sake of completeness.

3.5.1 Outline

As in the previous section: The Prover is sent C_N , an arithmetic circuit over $\mathbb{Z}_N$, and some input, x , for evaluation. Where this scheme differs from the previous is that the Verifier arbitrarily selects only one of the products p of N from P_N with which to independently evaluates the circuit $C_p(x)$. The result of this evaluation is then compared to the result provided by the Prover. The Verifier's result should be congruent to the Prover's result when reduced to the same modulus.

The ability to efficiently falsify a result that is still congruent to the correct result modulo an element of P_N , reduces to the ability to efficiently factor N . Assuming that N is sufficiently hard to factor, this construction is secure.

3.5.2 Protocol Definition

In order to better reason about the relative security of this construction, the instantiation of a suitably hard integer factoring problem is abstracted to an oracle, $\text{PrimeFactorProblem}(\lambda)$.

The Verifier generates a set of random co-prime numbers $P_N = \{p_1, p_2, \dots\}$, and their product N , via the integer factoring problem oracle $\text{PrimeFactorProblem}(\lambda)$. P_N is kept secret, and N can be freely shared.

$$(N, P_N) \leftarrow \text{PrimeFactorProblem}(\lambda) \quad (3.19)$$

The generation of N need only be done once and can be shared between verifications of different arithmetic circuits.

N is used to define the ring $\mathbb{Z}_N$ of integers modulo N . The Verifier notes that their arithmetic circuit C should be evaluated in $\mathbb{Z}_N$, subsequently denoted as C_N . Next, the Verifier bundles the arithmetic circuit C_N and the input data x into a tuple σ and sends it to the Prover for evaluation.

$$\sigma \leftarrow (C_N, x) \quad (3.20)$$

The Prover extracts C_N and x from σ , and calculates the result, r , of the evaluation of $C_N(x)$, then sends r back to the Verifier.

$$(C_N, x) \leftarrow \sigma \quad (3.21)$$

$$r \leftarrow C_N(x) \quad (3.22)$$

The Verifier selects an arbitrary $p \in P_N$ and independently evaluates $C_p(x)$.

$$r_p \leftarrow C_p(x) \quad (3.23)$$

The Verifier checks the congruence $r \equiv r_p \pmod{p}$ and if true, accepts the result from the Prover.

This procedure is expressed as definitions for the standard verifiable computation functions in Figure 3.9. Interestingly, a great deal of the work done by the Verifier is not dependent on the output of the Prover, and therefore can be executed asynchronously. The asynchronous exchange between Prover and Verifier is shown diagrammatically in Figure 3.10.

3.6 Evaluation

In this section properties for the improved VC protocol relating to security, computational complexity, and bandwidth complexity are proved.

3.6.1 Completeness

Theorem 3.6.1 (Completeness). An honest Prover can convince the Verifier of a correct result r to the computation $C_N(x)$ with probability

$$\Pr[\mathbf{Vf}(r) = \top] = 1 \quad (3.24)$$

KGen (λ) 1 : $(N, P_N) \leftarrow \text{PrimeFactorProblem}(\lambda)$ 2 : return (N, P_N)	ProbGen $_N(C, x)$ 1 : $\sigma \leftarrow (C_N, x)$ 2 : return σ	Compute (σ) 1 : $(C_N, x) \leftarrow \sigma$ 2 : return $C_N(x)$
---	---	---

Vf $_{P_N}(\sigma, r)$ 1 : $(C_N, x) \leftarrow \sigma$ 2 : $p \in P_N$ 3 : $r_p \leftarrow C_p(x)$ 4 : if $r \equiv r_p \bmod p$ then 5 : return $\top$ 6 : endif 7 : return $\perp$
--

Figure 3.9: The KGen, ProbGen, Compute, and Vf function definitions for the VC scheme, where KGen has access to an oracle capable of formulating the integer factoring problem for a given security parameter, PrimeFactorProblem(λ).

VC from the Hardness of Integer Factoring

Verifier	Prover
$(N, P_N) \leftarrow \text{PrimeFactorProblem}(\lambda)$ $\sigma \leftarrow (C_N, x)$	
	$\xrightarrow{\sigma}$
$p \in P_N$ $r_p \leftarrow C_p(x)$	$(C_N, x) \leftarrow \sigma$ $r \leftarrow C_N(x)$
	$\xleftarrow{r}$
if $r \equiv r_p \bmod p$ then return $\top$ endif return $\perp$	

Figure 3.10: VC from the hardness of integer factoring protocol overview. This can be derived by applying the function definitions in Figure 3.9 to the exchange detailed in Figure 4.6 with some of the Vf functionality precalculated by the Verifier as they await r from the Prover.

The completeness of this scheme can easily be derived by assuming both the Prover and Verifier act honestly and conform to the protocol.

Proof. An honest Prover will produce r , such that the r is the correct output of `Compute`.

$$\sigma \leftarrow \text{ProbGen}(C_N, x) \quad (3.25)$$

$$r \leftarrow \text{Compute}(\sigma) \quad (3.26)$$

As per the definition of `Compute` in Figure 3.9

$$r = C_N(x) \quad (3.27)$$

An alternative to calculating r as the output of $C_N(x)$ is to translate x into a residue number representation, calculate each residue independently, and then reconstitute them into r . The Chinese remainder theorem states that a value has one and only one decomposition into a given RNS [43].

The Verifier can select any $p \in P_N$ to produce a single residue of r , r_p .

$$r_p = C_p(x) \quad (3.28)$$

As r was calculated correctly, and it has only one representation in an RNS, it must be the case that it is congruent to each of its partial representations when reduced to the same modulus.

$$r \equiv r_p \pmod{p}, \forall p \in P_N \quad (3.29)$$

Therefore, when interacting with an honest Prover, no matter which element $p \in P_N$ the Verifier selects, $r \equiv r_p \pmod{p}$ will always be true and so `Vf` will always return $\top$.

$$\therefore \Pr[\text{Vf}(r) = \top] = 1 \quad (3.30)$$

□

3.6.2 Soundness

The *Soundness* of this scheme is based on the fact that fabricating a fake result $r' \neq r$ with a non-negligible probability of being rejected during `Vf`, hereafter referred to as a *viable forgery*, is as hard as decomposing N into P_N . It is proven later in this subsection that a viable forgery $r' \neq r$ with any possibility of going undetected during `Vf` can be used to extract non-trivial factors of N . Proving that generating a viable forgery is at least as hard as factoring N .

The exception to this reasoning is in the case that r' is undetectable under every element of P_N . The following Lemma proves this causes a contradiction and is not possible when $r \neq r'$.

Lemma 3.6.2. No adversary can exist with advantage of 1 in the `VerifyGame`, against this VC scheme.

$$\text{Adv}_{\mathcal{A}}^{\text{verify}}(\lambda) \neq 1 \quad (3.31)$$

Proof. Assume an adversary does exist that can create a viable forgery $r' \neq r$ such that it has $\text{Adv}_{\mathcal{A}}^{\text{verify}}(\lambda) = 1$. For this forgery it must be the case that under all elements $p \in P_N$, any of which could be chosen during Vf , r' is congruent to r when reduced by p .

$$\forall p \in P_N, r \equiv r' \pmod{p} \quad (3.32)$$

As per the Chinese remainder theorem, for each integer $r \leq N$ there is a unique decomposition into residual parts r_p [43]. Thus, if it is the case that for every element of $p \in P_N$, $r \equiv r'$ it must be the case that:

$$r \equiv r' \pmod{N} \quad (3.33)$$

The range of the function `Compute` is $[0, N)$, so any viable forgery must also be in this range. This fact, coupled with Equation 3.33 indicates the forgery $r' = r$. Therefore the adversary is incapable of producing a forgery, and such an adversary cannot exist without causing this contradiction. $\square$

Theorem 3.6.3 (Soundness). The advantage of a computationally bounded adversary in the `VerifyGame`, $\text{Adv}_{\mathcal{A}}^{\text{vf}}(\lambda)$ is at most that of the advantage of a computationally bounded adversary in the `FactorGame`, $\text{Adv}_{\mathcal{B}}^{\text{factor}}(\lambda)$.

$$\text{Adv}_{\mathcal{A}}^{\text{vf}}(\lambda) \leq \text{Adv}_{\mathcal{B}}^{\text{factor}}(\lambda) \quad (3.34)$$

The proof that follows is an extended version of the *Soundness* proof given in Section 3.4 for the previous construction. The proof is given by a reduction from forging results against this scheme to the integer factoring problem. First it is shown how one or more factors of N can be extracted from any viable forgery. Second, the ability to extract factors of N from r' is used to construct an efficient adversary for the integer factoring problem, given an adversary capable of efficiently making viable forgeries.

Proof. For a given computation $C_N(x)$ let r and r' indicate the correct result and an incorrect forged result respectively. A hypothetical Prover has submitted r' to the Verifier, claiming it is the correct output of the computation. When executing `Vf` the Verifier will calculate $r_p \leftarrow C_p(x)$ and check the congruence $r' \equiv r_p \pmod{p}$ for an element $p \in P_N$. As r_p is calculated independently by the Verifier, it must be the case that $r_p \equiv r \pmod{p}$. So during `Vf` the Verifier is effectively checking that $r' \equiv r \pmod{p}$.

In order for it to be at all possible that the Verifier may erroneously approve r' , it must be the case that r' is congruent to r when reduced modulo at least one of the factors of N . More formally:

$$\exists p \in P_N, r' \equiv r \pmod{p} \quad (3.35)$$

Consider a general adversary, capable of generating a viable forgery r' , such that it is undetectable under one or more element of P_N . In such a case there exists a non-empty set $Q \subset P_N$ and positive integer c where:

$$\forall p \in Q, r \equiv r' \pmod{p} \quad (3.36)$$

$$r = r' + c \prod_{p \in Q} p \quad (3.37)$$

$$|r - r'| = c \prod_{p \in Q} p \quad (3.38)$$

Euclid's Greatest Common Divisor (GCD) algorithm can then be used to extract $\prod_{q \in Q} q$, a non-trivial factor of N .

The following reduction shows how to construct an efficient adversary to the integer factoring problem, given an efficient adversary against this VC scheme, $\mathcal{A}$. The interface to $\mathcal{A}$ is identical to that previously given in Figure 3.7 and, thanks to Lemma 3.6.2, is capable of producing forged result with advantage

$$\text{negl}(\lambda) < \text{Adv}_{\mathcal{A}}^{\text{vf}}(\lambda) < 1 \quad (3.39)$$

given the public parameters C and N . The above inequality represents bounds any viable adversary to this scheme, thanks to Lemma 3.6.2, it is safe to assume $\text{Adv}_{\mathcal{A}}^{\text{vf}}(\lambda) \neq 1$, and therefore $Q \neq P_N$.

Given adversary $\mathcal{A}$, it is straightforward to construct adversary $\mathcal{B}$, identical to the one previously outlined in Figure 3.8. $\mathcal{B}$, with a single query to $\mathcal{A}$, can efficiently recover two non-trivial factors of N . In detail, when provided with a number N to factorise, $\mathcal{B}$ chooses a suitable arbitrary arithmetic circuit C from the space of possible circuits F . The circuit chosen does not matter, so it is in the best interests of the adversary to select a simple one. C and N are passed into adversary $\mathcal{A}$ which returns σ and r' . x is extracted from σ and used to calculate $r \leftarrow C_N(x)$ from which two non-trivial factors of N are extracted by finding the greatest common divisor of N and $|r - r'|$. A complete list of prime factors can then be found with recursive calls to adversary $\mathcal{B}$.

Using this construction, when provided with any viable adversary to the VC scheme, it is possible to construct an adversary in the integer factoring game with advantage $\text{Adv}_{\mathcal{B}}^{\text{factor}}(\lambda) = 1$. It follows that

$$\text{negl}(\lambda) < \text{Adv}_{\mathcal{A}}^{\text{verify}}(\lambda) \Rightarrow \text{Adv}_{\mathcal{B}}^{\text{factor}}(\lambda) = 1 \quad (3.40)$$

$$\therefore \text{Adv}_{\mathcal{A}}^{\text{verify}}(\lambda) \leq \text{Adv}_{\mathcal{B}}^{\text{factor}}(\lambda) \quad (3.41)$$

□

It is also worth noting that this scheme can detect non-malicious spontaneous errors in remote computation with probability

$$\Pr[\text{Detect Error}] = 1 - \Pr[r' \equiv r \pmod{p}] \quad (3.42)$$

However, importantly, for any adversarial strategy the adversary can do no better than an adversarial strategy against the integer factoring problem. This includes tactics such as a number field sieve [31], or randomly guessing prime number factors of N from which to base a forged r' .

3.6.3 Outsourceability

This section considers the computational complexity of this VC scheme and define the situations in which it is *Outsourceable*.

Prover

As per the definition of **Compute**, the Prover executes programs as if they were not being verified, without any change, and so has complexity $\mathcal{O}(|C_N|)$. It is worth noting here that related schemes require a Prover does an amount of work that is a multiplicative factor of the effort required to evaluate circuit C . In this scheme the Prover need only evaluate the unmodified target circuit C_N and so does less computational work than in the related work. For computations natively expressed over C_N this is optimal.

Verifier

The one-time setup cost for this scheme involves generating an instance of the integer factoring problem. Assuming suitable N s are not overly sparse this step has computational complexity $\mathcal{O}(|P_N|)$. This is derived from making a linear number of calls to oracle $\text{PrimeFactor}(\lambda)$, until a suitable N is generated. The primary cost being the generation of prime numbers and their subsequent multiplication together.

In this VC scheme **ProbGen** does nothing except pack the arithmetic circuit C_N , and the input data x into a tuple. This is only done to conform with the related work, and in reality this step can be omitted all together. Therefore, this step has a computational cost of 0.

The more sizeable contribution to the computational complexity of the Verifier is the **Vf** function, which has computational cost $\mathcal{O}(|C_p|)$. In the worst case, for N with overly large factors, this naively duplicates the effort of the Prover. Any improvement over this worst-case is obtained from a reduction in computational complexity when operating over smaller operands, as would be the case when operating $\text{mod } p$ as compared with the larger operands of the same computation $\text{mod } N$.

Theorem 3.6.4 (Outsourceable). For computations with intermediate operands larger than the factors in an underlying instantiation of the integer factoring problem, the computational complexity for the Verifier is less than than the computational complexity of evaluating arithmetic circuit C_N .

The proof is trivial given the assumption that the intermediate operands are large, and with the Verifier evaluating C_p , their computation will have smaller intermediate operands than C_N . It is worth noting here that being *Outsourceable* for only specific computational contexts is perfectly acceptable within the related work, much of which requires repeat executions to amortise a heavy pre-processing step in order to be considered *Outsourceable*. Or is only *Outsourceable* for specifically structured circuits.

A conservative estimation can be made of the expected performance of this VC scheme when verifying computations featuring intermediate operands uniformly distributed in $\mathbb{Z}_N$ by considering the prevalence of RNS arithmetic acceleration. RNS acceleration is a common practice for accelerating the evaluation of arithmetic circuits over $\mathbb{Z}_N$, and involves decomposing the circuit into $|P_N|$ circuits, each over an element of $p \in P_N$ and evaluating each separately [107]. The prevalence of this acceleration technique suggests that for computations with uniformly distributed intermediate operands in $\mathbb{Z}_N$, the computational complexity of an RNS must be no worse than computing the same operation in a more conventional number system. This conservative upper limit on RNS computational complexity can be used to create a average case computational complexity for **Vf** as $\frac{1}{|P_N|}$ the computational cost to evaluate C_N .

3.6.4 Performance Implications of the Factoring Problem

The belief that integer factoring is computationally intractable is a standard cryptographic assumption. However, algorithms for factoring integers are continuously being developed with incremental gains in factoring performance. Because of this ongoing progress references to the underlying factoring problem are kept intentionally abstract, and the problem generation is removed to an oracle. This abstraction protects the scheme from any minor developments in integer factoring.

That being said, the properties of the factoring problem have a direct impact upon this VC scheme. As shown in Theorem 3.6.4, as long as N has more than one factor this VC scheme is *Outsourceable* for computations with large intermediate operands, and the amount of work done by the Verifier shrinks as the number of factors of N increases. This relies on the fact that the individual complexity of evaluating the arithmetic gates is simpler if the operands are smaller. The gain in computing C_p over C_N is only valuable

if p is substantially smaller than the expected intermediate results. Otherwise, it is identical to calculating C_N . In which case this scheme is no better than naively double checking the result.

The difficulty of a given instance of the integer factoring problem is largely dictated by the size of N 's smallest factors. The smaller the smallest factors of N are the easier it is to find N 's factors. In order to speculate on the real-world performance of the system, consider the RSA cryptosystem. RSA is built on the same assumption as the scheme, the computational intractability of integer factoring. In RSA N is thousands of bits long, and, any p must be many hundreds of bits long [91]. For many conventional computations evaluating C_N and C_p is identical.

There is an inherent tension between selecting a performant N , that has multiple smaller factors, and a more secure N , that has fewer larger factors. Depending on the context the decision on how to reconcile these two factors may differ. To achieve a minimum degree of security, as given by the closely related and well-studied security implications of the RSA problem, the Verifier should expect to evaluate a computation $\text{mod } p$ where p is many hundreds of bits long. For some problems, namely Homomorphic Encryption (HE) evaluation as is explored later in this thesis, this can still be a significant improvement over local-only evaluation.

3.6.5 Proof Size

As compared with unverified computation, the scheme requires no extra communication. The only messages passed between the Prover and the Verifier are a description of the arithmetic circuit, C_N , the input, x , and the result, r . As such nothing extra is sent as compared with unverified computation. The total communication bandwidth is:

$$\mathcal{O}(|C| + \log x + \log r) \quad (3.43)$$

This leads to an interesting quality of the scheme; because the workload sent to the Verifier is unchanged, when compared with unverified computation, there is no reason that the Verifier would know they are engaged in a VC scheme. The Prover is effectively blind to their participation in a VC scheme.

To the best of my knowledge this is the first verifiable computation scheme to not require a witness of result correctness - it is effectively proofless.

3.6.6 Adapting General Problems

An obvious question arises: What about calculations that do not cleanly conform to the requirements on N ? It is known that arithmetic circuits are a good model for general computation, but there is a possibility the program of interest cannot be expressed as an arithmetic circuit over a ring suitable for the scheme.

3.6.7 $\mathbb{Z}_N$ Too Large

As previously discussed the degree of *Outsourceable*-ness of this VC scheme is based on the size of the factors of N and the intermediate operands of the target computation. The worse case scenario, where the factors of N are much larger than the intermediate operands, results in the Verifier doing exactly as much work as if they naively duplicated the computation. Importantly this is no *worse* than if they naively duplicated the computation, and so the *Outsourceable* property is maintained. But, any gains for the Verifier in terms of computational cost are lost when the ring is too large.

3.6.8 $\mathbb{Z}_N$ Too Small

It can be the case that the natural N for a target computation is extremely large, much larger than would be needed to make the VC scheme secure. In fact such a large N could make evaluation problematic for the Prover. In which case N can be easily decomposed into two, or more, sub-problems. This decomposition can be done by splitting the set P_N into exclusive subsets P_{N1} and P_{N2} and evaluating the arithmetic circuit as two independent problems, then recombining the results. This is a more standard application of an RNS. Given that N is overly large it is likely the case that $N1$ and $N2$ are also suitably hard to factor and so the scheme is still secure. The way to think about this is that an RNS is used to independently reduce the problem size from N into $N1$ and $N2$, and then the VC detailed in this work is used to outsource and verify the evaluation of C_{N1} and C_{N2} . Then reversing the first RNS to reconstitute the result mod N .

3.7 Conclusion

From Table 3.2 the biggest improvements of the VC scheme in this work over the related work are the massively reduced initial setup, and the proof size of zero. It forgoes some nice secondary properties present in related work; such as program, input, or output secrecy; zero knowledge; or public verification; but the scheme in this work has four interesting properties in its own right.

1. Prover blindness.
2. Proper optimal computational cost.
3. Extremely low overall latency for round trip computation and verification.
4. Implementation ease.
5. Program path verification.

First, the Prover is blind to the fact the computation is being verified as part of a VC scheme. This is because no additional information passes between the Prover and Verifier than would otherwise occur when engaged in a non-verified remote computation protocol. As such this scheme is completely proofless.

Second, as Prover operates on unmodified arithmetic circuits, it is impossible for them to do any less work without reimagining the scheme to operate over some abstract model of computation more efficient than arithmetic circuits. Therefore, this scheme is Prover-optimal in the arithmetic circuit setting.

Third, the scheme has extreme low latency from formulating C to receiving and verifying a result. The evaluation of $C_p(x)$ done by the Verifier during **Vf** can be executed before r is received from the Prover. This approach is shown in Figure 3.10. As such, the time taken to output a verified result is the negligible cost of **ProbGen**, plus **Compute**, plus the time taken to perform a single modular reduction by p . It is hard to imagine a VC protocol which could take less time from start to finish. This is in comparison to the related work where the Verifier must await the results and a witness for the correctness of the result before engaging in **Vf**.

Fourth, this scheme is relatively easy to implement. An existing client/server computation system only need implement **KGen** and **Vf**, as **Compute** is unchanged and **ProbGen** only packs data into a tuple in order to better align with related work.

Finally, the scheme has an additional interesting property in that the path taken through the program is verified. Should a result be accepted during **Vf**, the Verifier will have executed the same sequence of

	ProbGen	Compute	Vf	Proof Size	Complexity Assumption
Pinocchio [95]	$\mathcal{O}(C)$	$\mathcal{O}(C)$	$\mathcal{O}(\log x + \log^r)$	$\mathcal{O}(1)$	q-PKE [67], q-PDK [67], q-SDH [26, 57]
[100]	$\mathcal{O}(B)$	$\mathcal{O}(B)$	$\mathcal{O}(\log x + \log B)$	$\mathcal{O}(\log B)$	Low Complexity Cryptographic Hash Functions [10]
Bulletproofs [34]	0	$\mathcal{O}(C)$	$\mathcal{O}(C)$	$\mathcal{O}(\log C)$	DLP [86]
Libra [111]	$\mathcal{O}(\log x)$	$\mathcal{O}(C)$	$\mathcal{O}(d \log C)$	$\mathcal{O}(d \log C)$	q-SBDH, eq-PKE
No VC (Insecure)	0	$\mathcal{O}(C)$	0	0	
Naive Duplication	0	$\mathcal{O}(C)$	$\mathcal{O}(C)$	0	
This	0	$\mathcal{O}(C_N)$	$\mathcal{O}(C_p)$	0	Integer Factoring

Table 3.2: Related work Table 3.1 extended to include the new VC scheme presented in this work.

operations as the Prover. This means that the Verifier can independently confirm any computational costs levied by the Prover. This property could be very useful for verifying the billing costs from a cloud provider.

This chapter described the first VC construction built from the intractability of factoring integers. This construction only used standard cryptographic assumptions and, as per Theorems 3.6.1, 3.6.3, and 3.6.4 the scheme presented in this work has the *Completeness* and *Soundness* properties and is *Outsourceable* for computations with large intermediate operands. $\mathbf{Vf}$ is more expensive than in related work but unlike in related work this scheme has a small one-time setup independent of the computation. Despite the more expensive $\mathbf{Vf}$ the setup cost savings across the whole scheme result in a scheme that is never worse than naive duplication plus a single modular reduction and integer comparison, even for only a single execution. Unfortunately, the degree of outsourceableness of the scheme is intrinsically linked to the expected operand size of the programs being verified and the security of the scheme. In the worst possible case the Verifier does as much work as if they were duplicating the work to check the program output. As such the best performance of the system is when operating with large integer operands much larger than p .

While other schemes have better asymptotic properties, it is plausible that this scheme has better computational cost when verifying low-volume programs due to the zero cost setup phase. The scheme provides some novel properties, such as prooflessness, Prover-optimality, low latency, and relative implementation ease.

Chapter 4

Two Party Fair Exchange

In the design of secure protocols it is often the case that two mutually distrusting parties wish to exchange some data. Both parties only wish to reveal their data contingent on a guarantee that the other party will also reveal their data. This deadlock can be seen when devices wish to exchange digital goods for currency, swap secret information, or any other number of digital transactions. If one or more of the parties is untrusted, possibly because they are anonymous and there is no method of recourse to punish a cheater, parties will be hesitant to go first for fear of being cheated.

Fair Exchange (FE) schemes are a class of cryptographic construction facilitating the exchange of digital goods between two mutually distrusting parties. When participating in a FE scheme either both parties receive each others' secret data, or the protocol terminates with no data changing hands.

The work presented later in this thesis, in Chapter 5, makes black-box use of two party FE protocols to support the exchange of tokenised credit rewards for the results of outsourced computations. If the token is sent first, a malicious node performing the outsourced computation need never attempt the computation in the first place. If the result of the outsourced computation is sent first, a malicious work-issuing node may never send the token as payment for the work. To this end FE is required to ensure that all stages of the exchange are observed, even if nodes are predisposed to cheating. Because the protocols presented later in this thesis are decentralised by design, all FE protocols requiring a trusted third party to perform arbitration are fundamentally unworkable.

To the best of the author's knowledge all previous FE constructions with strong soundness guarantees require a trusted third party to intervene at some point. In the best case the third party can be mostly offline and need only intervene in the case of a dispute. It is widely believed that constructing an FE protocol without a trusted third party is impossible [93]. The problem of two party FE is a core component to decentralised distributed computation protocols as distrusting nodes often wish to exchange digital goods between themselves. As these protocols are decentralised there can be no trusted third party administrative node to perform arbitration

This chapter details work culminating in the description of the first two party FE protocol with strong soundness guarantees. First, two intermediate constructions are defined that provide iterations over a toy protocol for the two party FE of single bit secrets. In the first intermediate construction the secret bit held by each party is randomly determined, and the second intermediate construction extends the first to allow for each party to select their secret bit. These constructions are explored and it is shown that the one-bit construction does not scale well to arbitrarily sized secrets. Then a final construction is defined, related in

concept to the intermediate constructions, which is able to support two party FE with arbitrary sized secrets. The work in this chapter is summarised in [49].

4.1 Introduction

FE schemes can be applied in any situation in which fair communication between two parties needs to occur as a single atomic transaction. The exchange of digital goods, as highlighted above, is a standard application, but functional FE protocols open up possibilities for whole classes of problems in the domains of MPC protocols, secret sharing schemes, and consensus to name a few.

Consensus is a fundamental problem in computer science in which a swarm of independent processes must agree on some piece of data. This data agreed on can be anything from a sequence of database mutations to electing an authoritative node. As shown in [93] two party FE can be used to construct consensus protocols.

The work in this chapter aims to remove the requirement for a trusted third party when engaging in an FE protocol. This would allow nodes in a decentralised network to exchange digital goods with each other with confidence. In the work presented in Chapter 5 of this thesis, two party FE is a prerequisite for the point-to-point credit system introduced to support “tit-for-tat” computation.

4.1.1 Contributions

This chapter provides:

- An interrogation of a widely cited impossibility result, to find instances of the FE problem in which the result doesn’t hold.
- Extensions for standard FE definitions in order to include computational security.
- An exploration of new ideas for candidate two party FE strategies.
- Definitions for four incremental two party FE schemes for which:
 - The first intermediate protocol only supports exchanging randomly generated single bit secrets against semi-honest adversaries.
 - The second intermediate protocol only supports exchanging user chosen single bit secrets against semi-honest adversaries.
 - The third protocol supports exchanging arbitrarily sized secrets against semi-honest adversaries.
 - The fourth and final protocol supports the exchange of arbitrarily sized secrets against arbitrarily malicious adversaries.
- Arguments that key protocols defined in this chapter satisfy the FE properties of *Computational Completeness* and *Computational Strong Fairness*.

4.1.2 Threat Model

This work targets pairs of participants with access to approximately equivalent computational power without access to a trusted third party. This is in some ways a weaker threat model than much of the related work, which is defined in the context of adversaries that are polynomially computationally bounded in some security

parameter, albeit with a trusted third party. Adversaries are attempting to learn some secret without offering up a secret of their own. Initially this work only considers semi-honest adversaries, who can only deviate from the protocol by acts of omission, and are unable to send dishonestly generated messages. This restriction is lifted later in the chapter.

4.1.3 Assumptions

This work makes use of the standard assumption that there exists an IND-CPA secure symmetric authenticated encryption scheme. This encryption scheme is used as a black box in the presented FE constructions, the interface to which is described in the Background section of this chapter.

Assumption 2. There exists an IND-CPA secure symmetric authenticated encryption scheme.

It is important to note here that none of the security guarantees provided by being “authenticated” are required for soundness. It is used to provide an easy way to differentiate between “correct” and “incorrect” encryption keys.

Assumption 3. There exists a 1-in- n oblivious transfer protocol.

Generalised constructions for 1-in- n oblivious transfer have been realised in [29].

A conceivable but non-standard assumption is used; that for any given context there exists a “secure” $(t, \frac{1}{M})$ -secure concrete choice of security parameter k and corresponding a “bruteforceable” $(t, 1)$ -secure concrete choice of security parameter k' such that the adversarial advantage against each is related by a constant factor M . Expressed more generally:

Assumption 4. For any given encryption scheme $\mathcal{E}$, concrete security parameter k , time constraint t , and constant M , there exists a corresponding concrete security parameter k' such that $\mathcal{E}(k)$ is $(t, \frac{\varepsilon}{M})$ -secure and $\mathcal{E}(k')$ is (t, ε) -secure for some ε .

This is a relatively straight forward assumption, stated for use in later proofs. To better conceptualise the relationship between k and k' given in Assumption 4, consider the following concrete example: Given an instantiation of a cryptosystem deemed bruteforceable on commodity computer hardware in a month, $\mathcal{E}(k')$, Assumption 4 states there exists a corresponding instantiation $\mathcal{E}(k)$ bruteforceable on commodity computer hardware in 1000 years. In this example $M = 12,000$, and importantly, is fixed, even if new algorithmic developments reduce the time taken to mount an attack. In such a situation k and k' will be adaptively adjusted to maintain their expected time-to-bruteforce of 1000 years and 1 month respectively, and M is clearly unaffected. It is easy to see that M is therefore not directly related to k or k' and is instead dependent on the relationship between the two. No matter how M is chosen, it is constant as k and k' adaptively grow in any given domain. It is worth noting here that security practitioners tend to aim for a more existential time-to-attack for “secure” instantiations, usually on the order of the heat death of the universe and unfortunately the protocol presented in this work cannot satisfy this notion of security. In FE the ultimate goal is for targets to eventually exchange secrets, so a key space attackable in 1000 years on commodity hardware could very well be considered adequately secure.

Supporting Assumption 4, and allowing both protocol participants to reason about a secure choice of k and k' , requires the additional assumption that the computational capabilities of one of the two participants is publicly and reliably known.

NINDCPAGame ₁ (λ, n)	
1 :	$S \leftarrow \emptyset$
2 :	$b \leftarrow \$ \{0, \dots, n-1\}$
3 :	$k \leftarrow \mathcal{E}.\text{KGen}(\lambda)$
4 :	$m_0, m_1 \leftarrow \mathcal{A}()$
5 :	for i in $\{0, \dots, n-1\}$
6 :	$S[i] \leftarrow \mathcal{E}.\text{Enc}_k(m_0)$
7 :	endfor
8 :	$S[b] \leftarrow \mathcal{E}.\text{Enc}_k(m_1)$
9 :	$b' \leftarrow \mathcal{A}(m_0, m_1, S)$
10 :	if $b = b'$ then return 1
11 :	else return 0

Figure 4.1: The NIND-CPA security game, in which an adversary must correctly identify the element in set S which corresponds to an encryption of m_1 , where all the other elements in set S are an encryption m_0 . There is an implicit state maintained between the two calls to adversary $\mathcal{A}$.

Assumption 5. The computational capabilities of party X are publicly known to a degree of accuracy such that both participants can equitably negotiate protocol parameters k and k' .

This assumption allows both FE scheme participants to use the value for X 's expected time-to-attack a given cryptosystem as a factor in negotiating parameters for their shared FE scheme.

One of the potential attack vectors against a scheme such as those presented in this paper involves taking a set of partial protocol transcripts and extracting a weak key from this set. Soundness in the face of this attack requires making a further non-standard assumption. Consider a variation on the IND-CPA security game in which an adversary produces two messages, m_0, m_1 and is provided with an ordered set of size n where every element of the set is an encryption of m_0 except for a single element at a random index which is instead an encryption of m_1 . The following assumption states that there is no better way to attack this problem than applying an adversary against the standard IND-CPA security game to each element in turn until an encryption of m_0 is found. This game is given more formally in Figure 4.1.

Assumption 6 (NIND-CPA Security). The advantage of an adversary $\mathcal{A}$ in the NIND-CPA security game, detailed in Figure 4.1, is defined as:

$$\text{Adv}_{\mathcal{A}}^{\text{nind-cpa}}(\lambda, n) = \Pr[\text{NINDCPAGame}(\lambda) = 1] \quad (4.1)$$

There is no better algorithm to attack the NINDCPAGame than applying an adversarial subroutine against the IND-CPA security game, $\mathcal{B}$, to each successive element of set S in turn until an encryption of m_1 is found. Therefore:

$$\text{Adv}_{\mathcal{A}}^{\text{nind-cpa}}(\lambda, n) = (\text{Adv}_{\mathcal{B}}^{\text{ind-cpa}}(\lambda))^{\frac{n}{2}} \quad (4.2)$$

The following assumption allows participants in these FE schemes to know if and when the other participant disconnects early.

Assumption 7 (Witnessing Inaction of Participants). Participants in this FE scheme can witness the early disconnection of the other protocol participants.

In practice this assumption can be enforced with time constraints on messages, and as such the protocol presented in this work requires a synchronous network.

4.1.4 Additional Assumption for Initial Protocols

For the initial protocols presented in this work it is assumed that, except for terminating early, all parties will follow the protocol honestly.

Assumption 8. The protocol will be followed honestly, the only deviations allowed by an adversary are deviations by omission, in which an adversary can terminate early.

4.2 Background

As previously mentioned, the cryptographic building blocks used in this work are used in a black-box manner, as such references to these primitives are kept intentionally abstract. Additional notation and the required properties of any cryptographic building blocks are detailed in this section.

4.2.1 Notation

Alongside asymptotic notions of security, where relevant concrete security parameters are also used, the choice of a specific concrete security parameter is denoted with k . When dealing with concrete security, a scheme is (t, ε) -secure if an adversary bound in time t has, at most, probability ε of breaking the scheme. $\mathcal{E}(k)$ is used as shorthand to indicate the encryption scheme $\mathcal{E}$ used with security parameter k during KGen.

In pseudocode $\leftarrow$ indicates assignment, and $\leftarrow \$$ indicates an element is randomly sampled from a set. At times vectors are used to indicate an ordered set of data, these are denoted by a variable in bold $\mathbf{v}$. The i th element of the vector $\mathbf{v}$ is denoted by $\mathbf{v}[i]$. $[x; n]$ is used as shorthand to denote a vector of length n where each element is set to x . The concatenation of two ordered sets is given by $A = B \parallel C$.

The two participants in these FE schemes are referenced as party X and party Y . Variables associated with either party are usually subscripted with the appropriate letter. In the protocol description “ $X :$ ” is a prefix used to indicate the operation is executed by party X . The event of party X learning secret d_Y is denoted with the shorthand $X(d_Y)$. At times γ and δ are used to represent a complete or partial protocol transcript and round transcript respectively. The set of all possible protocol transcripts is given by Γ , and similarly Δ is the set of all possible round transcripts.

1-in- n oblivious transfer [29] is used as a black-box by the constructions in this chapter. The use of oblivious transfer is denoted with $a \xleftarrow{OT} \mathbf{y}[i]$, where a is the local variable used by the Receiver, $\mathbf{y}$ is an ordered vector of data stored on the Sender, and i is the index in $\mathbf{y}$ accessed as part of the oblivious transfer.

4.2.2 Fair Exchange

The following FE definitions are adapted from [79]. In an FE protocol party X holds one piece of data, d_X , while party Y holds a different piece of data, d_Y . Both parties wish to learn the data held by the other party. The protocol has the *Completeness* property if, with honest parties, X learns d_Y and Y learns d_X .

Definition 4.2.1 (Completeness). With honest parties X and Y , and respective data d_X and d_Y , when the protocol terminates X has learned d_Y and Y has learned d_X .

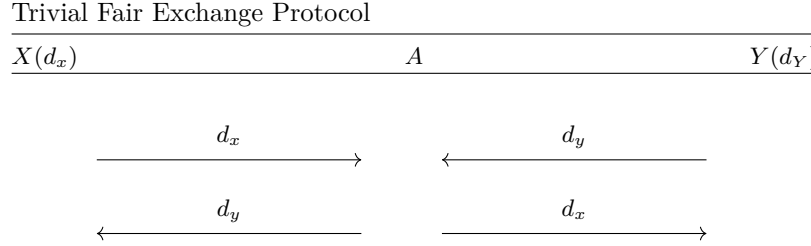


Figure 4.2: Trivial FE protocol with a trusted third party arbitrator, A .

The difficulty arises in constructing a protocol in which a dishonest participant is not able to discover the secret of the other participant without having to reveal their own secret. For example, there should be no way of X learning anything about d_Y without Y also learning d_X . The same is true for Y in relation to d_X . This property is *Strong Fairness*.

Definition 4.2.2 (Strong Fairness). When the protocol terminates, either X has learned d_Y or Y has gained no information about d_X . Similarly, when the protocol terminates either Y has learned d_X or X has gained no information about d_Y .

This definition is usually taken to imply perfect security, that no matter the computational capacity of either participant this soundness property holds. The *Strong Fairness* property contrasts with the type of fairness provided by early FE schemes. Earlier schemes usually operated by gradually disclosing a secret over many iterative rounds [82]. Obviously, with this approach a cheating participant can terminate at any point and have learned some information about the honest participant's secret. FE schemes featuring gradual disclosure, along with other related works, are discussed further in Section 6.5.

A naive solution to the problem of constructing a FE scheme with the *Strong Fairness* property is to allow all messages to pass through a trusted third party, this approach is shown in Figure 4.2. The third party validates the data from each party is as expected and then forwards the data to the interested parties. As mentioned in the assumptions laid out in the previous section, it is assumed all parties exchange data that satisfies some expected description. As such there is an implicit step in which all three parties agree to a description of the data being exchanged. This description is what the third party, A , validates before releasing data to either X or Y . The current state-of-the-art only requires the trusted third party to only intervene in the case of a dispute [79], which means the third party can be mostly offline and mediate disputes after the fact.

Impossibility Result

It is widely believed that *Strong Fairness* is impossible without the use of a trusted third party due to a proof that relates the FE problem to consensus [93]. In light of this result related works in this field often characterise the problem as impossible:

“Pagnia proved that it is impossible to construct a strong fair exchange protocol in the absence of a third party” [83]

“A third party is mandatory” [47]

“It is known that complete fair exchange [...] is not possible without a Trusted Third Party (TTP)” [101]

This belief stems from a result in which two party FE protocols with the *Strong Fairness* property can be used to construct consensus schemes. It has been shown that there can exist no deterministic algorithms for consensus in the asynchronous setting [55], therefore there can be no deterministic two party FE protocols in the asynchronous setting. Despite the impossibility of deterministic asynchronous consensus there is a rich history of usable consensus protocols which sidestep the impossibility result by relaxing the computational model - either by requiring network synchrony [96], or allowing a probabilistic algorithm with a possibility of non-termination [22]. Clearly, the impossibility result of [93] is misunderstood in the wider research community, being often described as asserting the absolute impossibility of two party FE. Given a closer reading of this work this is not the case, the work of [93] instead provides us with a reduced search space of possible two party FE protocols, any asynchronous deterministic algorithms can be immediately eliminated as fundamentally flawed.

The protocols presented in this chapter take a similar approach to solutions to consensus, targeting a probabilistic approach with a possibility of non-termination and some notion of network synchrony.

4.2.3 Symmetric IND-CPA Secure Authenticated Encryption Schemes

A symmetric encryption scheme of the form $\mathcal{E} = (\text{KGen}, \text{Enc}, \text{Dec})$ is an authenticated encryption scheme if it is able to provide ciphertext integrity and authenticity, along with the standard property of confidentiality of data. The standard symmetric encryption scheme function definitions, as originally provide in Section 2.4, are augmented with an extended Dec function of the form

- $\text{Dec}_k(c) \rightarrow m$ or $\perp$; decrypt ciphertext c to message m , with an invalid combination of c and k decrypting to $\perp$.

In the case a ciphertext has been tampered with in transit, or does not originate from the expected sender Dec outputs $\perp$. Authenticated encryption schemes are usually composed from encryption schemes and Message Authentication Codes (MACs).

A scheme is IND-CPA secure if it is indistinguishable under a chosen plaintext attack [21]. This definition is consistent with the concept of IND-CPA security laid out in Chapter 2. As a reminder, and with a minor extension to specify authenticated encryption schemes, consider the following definition.

Definition 4.2.3 (Symmetric IND-CPA Secure Authenticated Encryption Scheme). Consider symmetric authenticated encryption scheme $\mathcal{E} = (\text{KGen}, \text{Enc}, \text{Dec})$. The advantage of an adversary $\mathcal{A}$ is the IND-CCA security game, detailed in Figure 2.1, as:

$$\text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(\lambda) = 2 \Pr[\text{INDCPAGame}(\lambda) = 1] - 1 \quad (4.3)$$

A cryptosystem $\mathcal{E}$ is IND-CPA secure if for any probabilistic polynomial time adversary $\mathcal{A}$

$$\text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(\lambda) \leq \text{negl}(\lambda) \quad (4.4)$$

Importantly, none of the protocols presented in this chapter require the “authenticated” property of an Symmetric IND-CPA Secure Authenticated Encryption Scheme for any soundness guarantees. The “authen-

ticated” property is used to easily identify when an incorrect decryption key is used. It’s easily conceivable that “authenticated” property could be dropped in favour of more protocol complexity to detect this case.

4.2.4 Oblivious Transfer

Oblivious Transfer is a class of protocol which allows a Sender to offer up a vector of data to a Receiver who is able to access a single element. The Sender does not learn the element accessed by the Receiver, nor does the Receiver learn any of the unaccessed elements in the vector. The constructions in this chapter make black-box use of 1-in- n oblivious transfer [29], as such the specific mechanics are unimportant.

4.2.5 Non-Interactive Zero Knowledge Proofs (ZKPs)

A non-interactive ZKP $\mathcal{Z} = (\text{KGen}, \text{Prove}, \text{Verify})$ is a procedure in which a Prover certifies the truthfulness of some mathematical statement to a Verifier without revealing any information other than the truthfulness of the statement [54, 64]. Non-interactive ZKPs are one of the techniques used in the final protocol in this chapter to enforce message preconditions. This constraint on messages is used to bootstrap soundness against arbitrarily malicious adversaries from soundness against semi-honest adversaries. A non-interactive ZKPs is defined by the following functions:

- **Key Generation:** $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(\lambda)$; generate public and private key pair, used for verification and proving respectively.
- **Proving:** $p \leftarrow \text{Prove}_{\text{sk}}(s, w)$; generate a proof p attesting to statement s with witness w .
- **Verification:** $\top$ or $\perp \leftarrow \text{Verify}_{\text{pk}}(s, p)$; verify the correctness of proof p on statement s , returning $\top$ if correct, and $\perp$ if incorrect.

ZKPs must accept proven statements, reject unproven statements, and leak no knowledge beyond the truthfulness of the statement. These properties are known as *Completeness*, *Soundness*, and *Zero Knowledge*. A proof system is secure if an adversary attempting to forge a proof, and therefore circumvent *Soundness*, has advantage $\text{negl}(\lambda)$. The interested reader is directed to [66] for a comparison of modern zero knowledge proving systems.

4.2.6 Commitment Schemes

A commitment scheme $\mathcal{C} = \text{Commit}$ allows one party to produce a commitment for an arbitrary value, which can be shared with another party. The commitment both hides the underlying value, and binds the original party to their original value [90]. These properties are known as *Hiding* and *Binding* respectively. Commitment schemes are sometimes defined in terms of two functions *Commit* and *Reveal*, but often this can be simplified to a single definition:

- **Committing:** $c \leftarrow \text{Commit}(x, r)$; produce commitment c to value x with randomness r .

Where the commitment can be revealed by sharing r , and others can validate the commitment by replaying the *Commit* function on the original inputs.

It is commonly known that commitment schemes can not be both perfectly *Hiding* and perfectly *Binding*, this work assumes that any chosen commitment scheme is at the very least computationally *Hiding* and computationally *Binding*, and as such an adversary attempting to undermine either of these properties has advantage $\text{negl}(\lambda)$.

4.3 Related Work

Early FE protocols relied on the technique of gradual disclosure, in which individual bits of a secret are released over many rounds [23]. In other works mutual disclosure has been extended to include the single-bit case, by simulating a biased coin which slowly reveals information about a secret over many rounds. This approach has the unfortunate side effect that early termination may leave one party, by sheer bad luck, with a misrepresentative sample of coin flips. This was later refined to simulate the production of a shared biased coin which when flipped would land on “same” or “different”, thereby ensuring both participants had an identical sample of flips [82]. By definition, no gradual disclosure schemes are able to satisfy any version of the *Strong Fairness* property.

Another early solution to the FE problem was constructed based on the assumption that Y is able to witness the behaviour of X after seeing d_Y , and is therefore able to discern d_X based on the course of action X took [71]. Interestingly, this assumption is a less realistic ability than one required in this work, wherein party X is able to witness a party Y disconnecting after successfully extracting d_Y .

There is a body of work solving the FE problem with financial incentives to encourage honesty [74]. In such work a misbehaving party must forfeit previously posted financial assets. The advent of blockchain has allowed a decentralised ledger to stand in for a trusted third party [35, 51, 61, 73]. Ethereum smart contracts can encode the behaviour of a trusted third party, and release Ethereum tokens should certain conditions be met [32]. This is unfortunately very expensive to implement, relies on the security of a blockchain and the honesty of participating nodes, and is restricted to the exchange of cryptocurrency tokens.

All previous FE protocols with any notion of *Strong Fairness* use a trusted third party in some way. The previous state-of-the-art was *Optimistic* as it only required a third party to mediate in the case of a dispute [79].

4.4 Extended Definitions for Fair Exchange

As part of the work presented in this chapter definitions are provided for the modified FE properties of *Computational Completeness* and *Computational Strong Fairness*. These new properties extend corresponding well-established FE definitions to be more suitable domain for the work presented in this chapter. *Completeness* and *Strong Fairness* offer absolute properties in which an FE scheme is guaranteed to terminate successfully with honest participants and thwart cheating participants respectively, commonly this security domain is known as Statistical Security, or Perfect Security. The new properties in this section make the related claims but with reductions to computationally intractable problems. An FE scheme with *Computational Completeness* and *Computational Strong Fairness* as defined in this section has a high probability of terminating successfully with honest participants, and participants will find it no easier to cheat than perform some computationally intractable task. Although these notions are weaker than the standard modern properties they are far stronger than the historical FE soundness found in gradual disclosure schemes.

The complimentary FE notion of *Completeness* in the computational security domain is *Computational Completeness*.

Definition 4.4.1 (Computational Completeness). Given X with data d_X and Y with data d_Y , if both X and Y act honestly, X learns d_Y and Y learns d_X with high probability.

The definition for *Computational Strong Fairness* similarly corresponds to the FE notion of *Strong Fairness*.

FairnessGame ₁ (λ)	
1 :	$\delta \leftarrow \$ \Delta$
2 :	$d'_Q \leftarrow \mathcal{A}_P(\delta)$
3 :	if $d_Q = d'_Q$ then return 1
4 :	else return 0

Figure 4.3: Strong Fairness security game for an FE protocol. δ is a potentially partial protocol transcript randomly sampled from the space of possible transcripts Δ , and $\mathcal{A}_P$ is an adversary acting as party P . If the adversary is able to extract the data of the other participant, d_Q , they win. P and Q are used to allow the adversary to distinguish between the specific rolls X and Y have in the given FE schemes.

Definition 4.4.2 (Computational Strong Fairness). For arbitrary party P define the advantage in the fairness game, as defined in Figure 4.3, of adversarial party P as:

$$\text{Adv}_{\mathcal{A}_P}^{\text{fairness}}(\lambda) = \Pr[\text{FairnessGame}(\lambda) = 1] \quad (4.5)$$

For a given protocol transcript δ , if the ability of party Y to extract d_X from δ is computationally intractable, then for any probabilistic polynomial time adversary $\mathcal{A}_X$

$$\text{Adv}_{\mathcal{A}_X}^{\text{fairness}}(\lambda) \leq \text{negl}(\lambda) \quad (4.6)$$

Similarly, if the ability of X to extract d_Y from δ is computationally intractable, for any probabilistic polynomial time adversary $\mathcal{A}_Y$

$$\text{Adv}_{\mathcal{A}_Y}^{\text{fairness}}(\lambda) \leq \text{negl}(\lambda) \quad (4.7)$$

4.5 Candidate Approaches

This section is a brief discussion on potential approaches to the two party FE problem. These solutions were considered early in the development of the schemes presented in this work and serve as a backdrop for the work that follows. It is important to understand why some of these relatively intuitive high level protocol ideas are not well suited for solving the two party FE problem.

4.5.1 Onion Encryption

An initial idea is an extension of the notion of a progressive decryption, taking inspiration from the layers of encryption used in onion routing [62]. The idea involves alternating layers of encryption on a shared payload between X and Y . The parties alternate stripping layers of decryption from this object and at some point in this process the secrets of each party are revealed.

It should be the case that out the first party to receive a secret is randomised. It also must be the case that the second secret is revealed immediately after the first as to reduce the possibility of cheating by early termination. If the first party to receive a secret decides to cheat and terminates early the spurned party must be able to, with great effort, bruteforce the final layer of encryption to the unrevealed secret.

Unfortunately, if this is the case, why does the second party not simply bruteforce the transition before each decryption they provide, and then terminate early? In that case, cheated party, who should have been the first party to learn a secret, will need to be able to bruteforce this transition to also receive a

secret. This logic can be applied recursively to arrive at the conclusion that each state transition needs to be brute-forceable. In which case whoever holds the shared secret construction first will never reveal it to the other participant and instead will brute-force all the transitions in their own time.

The problem here is that each brute-forceable individual state transition combines linearly to produce a construction that is brute-forceable from the initial state.

4.5.2 Leaking Keys Through Position of Secrets

This leads a second idea: what if the individual transitions are not brute-forceable but instead the knowledge of where the secret is stored in the chain of states can be used to decrypt the secret of the cheating party? Both parties collaborate once again to produce some mathematical object. Then, both parties alternately process the construct. At some point in this chain of state transitions the secrets are leaked. The position at which the first secret is leaked can be used to extract the second secret.

For a construction of this nature to work for FE the secrets have to be revealed on sequential state transitions, so that a cheating party cannot pretend to have not received anything. If they wish to avoid revealing their secret their only option can be to terminate early. This would reveal the position at which they learned a secret and then allow the other party to extract the secret they were originally denied.

Unfortunately this falls into the same recursive argument trap of the previous construction. The party that would originally receive a secret second can preemptively act as they would if the first party disconnected, by pretending their secret was going to be revealed on the next state transition. If this information is enough to reveal their peer's secret, then they disconnect before they release their secret. In which case, in order to be fair, the scheme must also be designed such that this information can be used by the party that would originally have learned a secret first to extract that secret anyway. This argument can continue recursively until either party can extract all the information from the original state and as such they never have to reveal any information at all.

Both of the **Onion Encryption** and **Leaking Keys Through Position of Secrets** candidate approaches are poorly suited to the two party FE problem. This is in line with the impossibility result [93], as these are deterministic algorithms in an asynchronous setting.

4.5.3 Shared State

The next evolution of these ideas involves removing the onion encryption, and by relation the deterministic alternating state changes, and instead develop the idea of leaking the secret of a cheating party through the event of a disconnection. In order to achieve this non-determinism and light synchrony must be added to the procedure, such that a single shared state allows X and Y to exchange results, but a number of failure states leak no information whatsoever. In this way a cheating party that disconnects confirms the validity of the shared state, by virtue of the fact they are trying to escape with a secret without offering one up in return.

In effect this style of construction takes the lack of a message from either party to be confirmation that both participants are in a success state. From here the party that has been cheated out of a secret can extract the secret that was denied to them. This approach appears to have potential as a two party FE scheme. The following constructions are attempts to construct a concrete scheme which matches this high level description.

4.6 Sharing a Single Random Bit

As an introduction to more complete constructions in the sections later in this chapter, this section specifies an intermediate FE construction in which the two parties exchange a single secret bit, but that bit must be determined randomly at the start of each round. This toy construction has no real world utility, and serves as a simpler introduction to the two related constructions which are provided later in this chapter. This construction requires the additional assumption that adversaries will be semi-honest, as given in Assumption 8.

As discussed in the previous section, the key insight from which this construction is derived is that the lack of a message from the party that receives a secret first, is in itself a type of response. In this definition, and the definitions to follow, Y learns X 's secret first. X 's final decoding step accepts the empty string, ε , as a valid input. In order to do this, a protocol is constructed where there is a single success state from which both parties can derive the other's secret, and many failure states, from which nothing is learned. Therefore if Y disconnects early they must have reached a success state, and therefore X can derive Y 's secret. Importantly, X should not be able to distinguish the shared state from random until they receive ε in confirmation from Y . This is possible thanks to Assumption 7.

To demonstrate the fundamentals of more complete constructions, this toy construction is detailed for the simplified situation where both X and Y randomly generate a single bit secret at the beginning of the protocol, and regenerate the bit if errors require the protocol to be repeated. In the subsequent section it is shown how to extend this construction to a predetermined single bit chosen by each participant.

4.6.1 Description

The protocol operates over multiple successive rounds until the secret bits are successfully exchanged. At the start of each round both X and Y randomly generate a single bit which they wish to share with each other, d_X and d_Y respectively. They both generate additional random bits r_X and r_Y . X then generates 2 encryption keys for an IND-CPA secure authenticated encryption scheme $\mathcal{E}$:

$$X : k_0 \leftarrow \mathcal{E}.\text{KGen}(\lambda), k_1 \leftarrow \mathcal{E}.\text{KGen}(\lambda) \quad (4.8)$$

X then encrypts an arbitrary message, h , with encryption key k_0

$$X : c \leftarrow \mathcal{E}.\text{Enc}_{k_0}(h) \quad (4.9)$$

and sends this message to Y . X constructs a four element vector $\mathbf{v}$, with k_0 at index $2d_X + r_X$, and with k_1 in all other positions.

$$X : \mathbf{v} \leftarrow \begin{cases} k_0 & \text{if index} = 2d_X + r_X \\ k_1 & \text{otherwise} \end{cases} \quad (4.10)$$

Y learns X 's value $\mathbf{v}[2d_Y + r_Y]$ via oblivious transfer. Y can then attempt to decrypt the message c :

$$Y : h \text{ or } \perp \leftarrow \mathcal{E}.\text{Dec}_k(c) \quad (4.11)$$

Y will only receive the correct key if they accessed the correct element in X 's vector $\mathbf{v}$. Therefore, decryption

will succeed if and only if $d_X = d_Y$ and $r_X = r_Y$. If decryption succeeds Y learns X 's secret bit because d_X must be the same as their own, d_Y . If decryption fails Y will be unsure if this is because $d_X \neq d_Y$ or their randomness was different, $r_X \neq r_Y$. If the scheme did not randomly fail like this, Y would be able to learn d_X from a failure as well as a success, as the failure would imply $d_X = 1 - d_Y$.

Upon a decryption failure, Y sends X the key k_1 as proof. The two parties then begin the next round of the protocol. On a success Y can either let X know, in good faith, or disconnect entirely. If Y sends X nothing, ε , X can deduce that Y must have successfully decoded their secret. They can therefore assume that $d_X = d_Y$.

4.6.2 Evaluation

For brevity proofs of security are omitted for this intermediate construction. The intuition behind the *Strong Fairness* of the scheme is that the act of disconnecting early leaks the fact that Y 's decryption was successful to X . By virtue of the fact Y learned something, X can deduce that they must have chosen the same bit to share with each other. So, X learns Y 's secret too.

This intermediate protocol has a serious deficiency: The secret bit X and Y swap with each other has to be randomly generated at the start of the protocol round. If this were not the case if the secrets did not initially align then the protocol would fall into continuous failure loop. At which point either party could correlate successive failures with a high probability of them choosing different messages from each other.

4.7 Sharing a Single Chosen Bit

This section specifies a further intermediate FE scheme, extending the previous to allow both parties to choose their single-bit secret which can be used throughout repeated rounds of the protocol. Extra randomness needs to be added in order to mask a secret bit of each party's choosing, without leaking information between failed protocol executions. Both X and Y mask their chosen bit, d_X and d_Y , with a single bit h_X and h_Y , which is randomly sampled, and resampled each time the protocol is repeated. Unfortunately this scheme does not stratify the *Strong Fairness* property, as an adversarial party can mount a trivial attack with a better-than-negligible probability of success, $\Pr[X(d_Y)] = 0.625$. However, this intermediate protocol serves as a stepping stone towards the final construction. As with the previous construction, this construction requires the additional assumption that adversaries will be semi-honest, as given in Assumption 8.

4.7.1 Description

Parties X and Y select a bit they wish to exchange, d_X and d_Y respectively. This protocol then operates over successive rounds, until a success state is reached. The definition of a single round this FE scheme is given by the function definitions in Figure 4.4. These functions are then composed into a protocol as shown in Figure 4.5.

At the start of each round both parties randomly generate a hiding bit h_X and h_Y and random bit r_X and r_Y . Both h bits and r bits are specific to this protocol round, and should be resampled when restarting after a round failure. X calculates their round bit as $s_X \leftarrow d_X \oplus h_X$, and Y calculates their round bit similarly

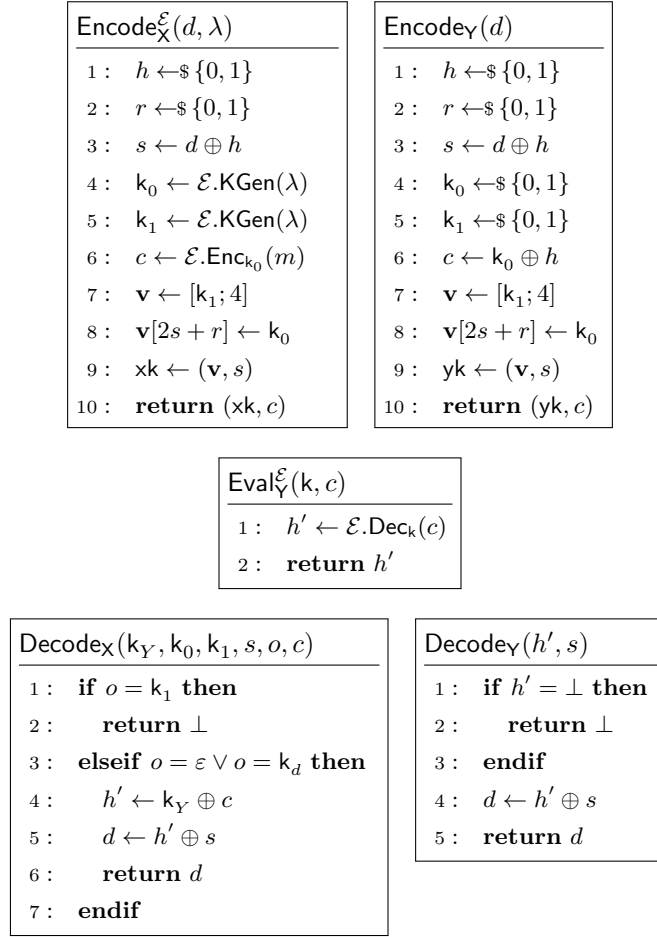


Figure 4.4: Function definitions for the chosen single-bit two party FE scheme. For notation simplicity it is assumed that variables are name-spaced to their own function.

One Round of the Two Party Fair Exchange with Chosen Single Bit Secrets

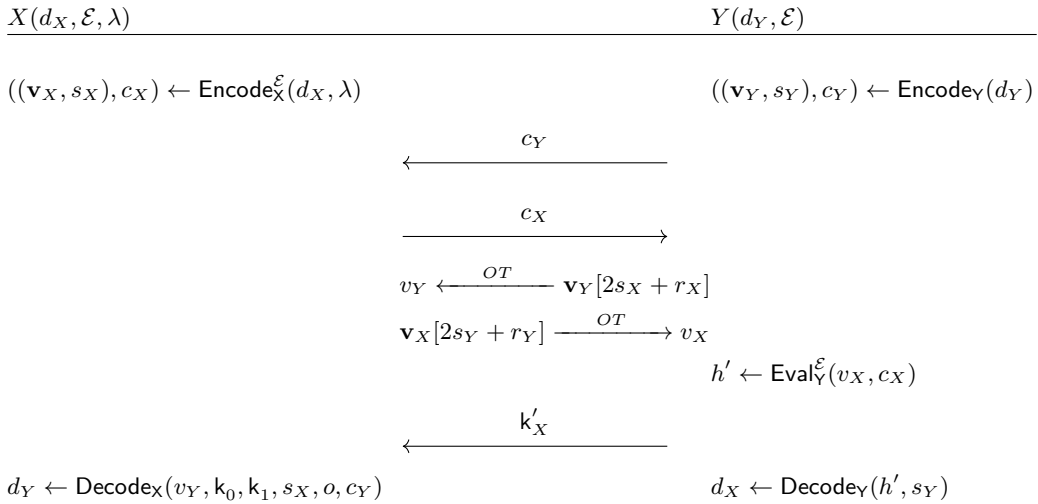


Figure 4.5: Two party FE protocol for chosen single-bit secrets.

as $s_Y \leftarrow d_Y \oplus h_Y$.

$$X : h_X \leftarrow \$\{0, 1\} \qquad Y : h_Y \leftarrow \$\{0, 1\} \qquad (4.12)$$

$$X : r_X \leftarrow \$\{0, 1\} \qquad Y : r_Y \leftarrow \$\{0, 1\} \qquad (4.13)$$

$$X : s_X \leftarrow d_X \oplus h_X \qquad Y : s_Y \leftarrow d_Y \oplus h_Y \qquad (4.14)$$

The encryption scheme used by Y is a one-bit one-time-pad and as such provides perfect security for one-bit messages. Keys for this one time pad are generated by Y sampling two random single-bit keys.

$$Y : k_{Y0} \leftarrow \$\{0, 1\} \qquad (4.15)$$

$$Y : k_{Y1} \leftarrow \$\{0, 1\} \qquad (4.16)$$

Y encrypts their hiding bit with key k_{Y0} arbitrarily chosen from the two randomly sampled keys.

$$Y : c_Y \leftarrow h_Y \oplus k_{Y0} \qquad (4.17)$$

and sends c_Y to X .

X generates two keys for IND-CPA secure symmetric authenticated encryption scheme $\mathcal{E}$:

$$X : k_{X0} \leftarrow \mathcal{E}.\text{KGen}(\lambda) \qquad (4.18)$$

$$X : k_{X1} \leftarrow \mathcal{E}.\text{KGen}(\lambda) \qquad (4.19)$$

X then encrypts their hiding bit h_X with arbitrarily chosen key k_{X0} :

$$X : c_X \leftarrow \mathcal{E}.\text{Enc}_{k_{X0}}(h_X) \qquad (4.20)$$

and sends this c_X to Y .

Y arranges their keys into a vector $\mathbf{v}_Y$ with the key they used in position $2s_Y + r_Y$ and the key they did not use in each other position:

$$Y : \mathbf{v}_Y \leftarrow \begin{cases} k_{Y0} & \text{if index} = 2s_Y + r_Y \\ k_{Y1} & \text{otherwise} \end{cases} \qquad (4.21)$$

Similarly, X constructs a four element vector $\mathbf{v}_X$, with the key they used k_{X0} , at index $2s_X + r_X$, and with they key they did not use, k_{X1} , in all other positions:

$$X : \mathbf{v}_X \leftarrow \begin{cases} k_{X0} & \text{if index} = 2s_X + r_X \\ k_{X1} & \text{otherwise} \end{cases} \qquad (4.22)$$

X learns the value of $\mathbf{v}_Y[2s_X + r_X]$ via oblivious transfer, X can then attempt to decrypt c_Y to learn h'_Y . If $s_X = s_Y$ and $r_X = r_Y$, then $h'_Y = h_Y$, otherwise h'_Y is just a randomly sampled bit.

$$X : h'_Y \leftarrow \mathbf{v}_Y[2s_X + r_X] \oplus c_Y \qquad (4.23)$$

Unfortunately for X , due to the perfect security of a one-time-pad encryption scheme they are unable to

distinguish between Y 's correct h_Y and uniform random until they observe Y 's behaviour after this point.

Next, Y learns the value of $\mathbf{v}_X[2s_Y + r_Y]$ via oblivious transfer. Y then attempts to decrypt the message c_X :

$$Y : h'_X \leftarrow \text{Dec}_{\mathbf{v}_X[2s_Y + r_Y]}(c_X) \quad (4.24)$$

As with X , the decryption will only succeed if $s_X = s_Y$ and $r_X = r_Y$, if this is the case Y learns X 's r value equals their own and, more importantly, X 's s value also equals their own. Y can then calculate X 's secret:

$$Y : d_X = h'_X \oplus s_X \quad (4.25)$$

From here, Y can act in good faith and let X know the decryption was a success, so they can extract Y 's secret similarly as $s_b = h'_b \oplus s_a$. But instead, if Y attempts to cheat and X receives nothing in response except the empty string, ε , X can safely assume the decryption was successful, infer that $s_X = s_Y$ and consequently extract Y 's d_Y in the exact same way as if Y acted in good faith.

Instead, if the decryption was a failure, Y sends X the key k_{X1} as proof. X receives this key, and recognises it as the key they didn't use. The two then regenerate hiding bits, h_X and h_Y , and random bits r_X and r_Y , and repeat the protocol until they achieve a success.

4.7.2 Attack Against Computational Strong Fairness

The intermediate protocol presented above does not satisfy the *Strong Fairness* property. An adversarial party X can mount the following attack and guess Y 's secret with probability $\Pr[X(d_Y)] = 0.625$:

1. Engage with the protocol honestly until the moment at k'_Y is received.
2. Assume k'_Y is the correct masking bit and output guess $d'_Y \leftarrow k'_Y \oplus s_X$.

A quarter of the time k'_Y is the correct masking bit, and so the guess is correct. When k'_Y is not the correct masking bit the correctness of the guess is left to random chance. Therefore the adversarial advantage when mounting this attack is $\Pr[X(d_Y)] = 0.625$. While this is the case for X , Y is unable to extract d_X with probability greater than $2\Pr[Y(d_X)] - 1 \leq \text{negl}(\lambda)$. Clearly this violates the upper bound on the contingent adversarial advantage of the two parties, as given by the definition of *Computational Strong Fairness* in Definition 4.4.2.

4.7.3 Evaluation

This scheme provides two party FE of secrets of a single bit in length. Intuitively, the scheme has the *Completeness* property but not quite the *Strong Fairness* properties.

Even ignoring the lack of *Strong Fairness*, this scheme does not map well to arbitrarily sized secrets. Unlike with other single-bit protocols this cannot be easily adapted through parallel executions, the common approach to adapting one-bit-version of protocols to arbitrary sized versions. In this case this approach does not work as the fairness of the scheme relies on knowledge about which individual instantiation of the protocol succeed or fail. In running many parallel invocations of the protocol Y would learn some of d_X on each iteration. But if the case Y has learned enough and attempts to cheat and replies with ε , X would have no way of knowing on which s and r bits aligned with Y and so could not extract d_Y . Therefore, extending this protocol naively to arbitrarily sized secrets degrades the *Fairness* property.

Similarly consecutive protocol invocations do not work either. This suffers from the same problem as when both parties reveal a single bit at a time. The difference between this and each party exchanging single bits at a time is that both parties discover each other's bit at the same moment. This minor improvement doesn't help the fact that the gradual disclosure of secrets fundamentally violates any notion of *Strong Fairness*.

Another attempt to modify this scheme for arbitrarily sized secrets involves increasing the number of failure states such that in a success state a secret key used by Y to encrypt a larger secret can be extracted. This could be achieved by having $2^\lambda - 1$ failure states, and a single failure state that implies a cryptographic key of length 2^λ . However, this is clearly infeasible as the expected number of times the protocol must be run to successfully exchange secrets grows exponentially in λ . In the following section a final scheme is presented which improves on this case, and allows for the exchange of secrets of arbitrary size after a configurable fixed expected number of protocol rounds.

4.8 Exchanging Secrets of Arbitrary Length

The FE scheme presented in this section adapts the high level approach of the single-bit schemes to the domain of arbitrarily sized secrets. As with the previous two constructions, this construction requires the additional assumption that adversaries will be semi-honest, as given in Assumption 8.

The naive approach to this adaptation involves expanding the number of failure states to allow the single success state to express the entirety of the secret key used by Y to encrypt their message. This poses two problems. The first of which was discussed in the previous section, in which the failure rate grows exponentially in λ . The second problem is that party X can mount an attack where, upon generating state s_X they can preemptively decrypt c_Y before allowing Y access to their vector of keys. This means that X would always have the opportunity to learn d_Y and stop Y learning d_X by preemptively disconnecting from the protocol.

The solution to this problem involves re-contextualising the schemes in terms of concrete security, as given in Section 4.2.1, and adding a second cryptosystem, $\mathcal{E}(k')$, related to the first, $\mathcal{E}(k)$, by way of the relative strength of the two systems. This second cryptosystem is brute-forceable in some long-but-reasonable time-frame, such as a month.

Recall Assumption 4: That there exists a constant M that defines the ratio between the difficulty of a secure-enough security parameter k and an insecure k' that is brute-forceable in some reasonable time-frame.

$$M \text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(k) = \text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(k') \quad (4.26)$$

By definition, attacking $\mathcal{E}(k)$ is infeasible. But if there are enough failure states in the FE scheme that if Y cheats X can learn which 1-in- M state both are in, and so the problem of breaking $\mathcal{E}(k)$ is reduced to the feasible problem of breaking $\mathcal{E}(k')$. With the addition of the second scheme once M has been set it is constant, even if k and k' are updated to keep up with advances in adversarial capabilities. This is because the ratio between the difficulties of the two schemes is constant.

The approach taken in this construction dramatically shrinks the number of failure states required to make the single success state meaningful enough for the fair exchange scheme to support arbitrary sized messages. This reduces the number of times the protocol must be run and also decouples the expected number of failures before a success from the security parameter k .

4.8.1 Description

Both parties negotiate a pair of concrete security parameters k and k' such that k is believed to be secure and k' should be brute-forceable in a long but reasonable time frame for party X given knowledge of their computational capabilities as per Assumption 5. $\mathcal{E}(k)$ is defined as $(t, \frac{\epsilon}{M})$ -secure and $\mathcal{E}(k')$ is defined as (t, ϵ) -secure for some large constant M , as per Assumption 4. Both participants now engage in an interactive multi-round protocol, continuing until the protocol successfully terminates.

At the onset of the round, X and Y each generate a key for the symmetric IND-CPA secure authenticated encryption scheme $\mathcal{E}(k)$. This key generation step must be repeated each round.

$$X : k_X \leftarrow \mathcal{E}.\text{KGen}(k) \qquad Y : k_Y \leftarrow \mathcal{E}.\text{KGen}(k) \quad (4.27)$$

They each then encrypt their respective data.

$$X : c_X \leftarrow \mathcal{E}.\text{Enc}_{k_X}(d_X) \qquad Y : c_Y \leftarrow \mathcal{E}.\text{Enc}_{k_Y}(d_Y) \quad (4.28)$$

Both parties freely exchange the generated ciphertexts. Party Y then generates a key for encryption scheme $\mathcal{E}(k')$, b . They use this value to encrypt their key, k_Y .

$$Y : b \leftarrow \mathcal{E}.\text{KGen}(k') \quad (4.29)$$

$$Y : c_{k_Y} \leftarrow \mathcal{E}.\text{Enc}_b(k_Y) \quad (4.30)$$

Both parties individually randomly sample m , such that $0 \leq m < M$, and construct large M -sized vector $\mathbf{v}$. X inserts k_X at position m_X in vector $\mathbf{v}_X$, and fills the other positions with random unused keys. Y inserts c_{k_Y} at position m_Y in vector $\mathbf{v}_Y$, and fills the remaining space with random irrelevant ciphertexts.

$$X : m_X \leftarrow \$\{0, \dots, M-1\} \qquad Y : m_Y \leftarrow \$\{0, \dots, M-1\} \quad (4.31)$$

$$X : \mathbf{v}_X \leftarrow \begin{cases} k_X & \text{at index } m_X \\ \mathcal{E}.\text{KGen}(k) & \text{otherwise} \end{cases} \quad (4.32)$$

$$Y : \mathbf{v}_Y \leftarrow \begin{cases} c_s & \text{at index } m_Y \\ \mathcal{E}.\text{Enc}_b(z) & \text{with randomness } z \text{ otherwise} \end{cases} \quad (4.33)$$

First, party X retrieves element $\mathbf{v}_Y[m_X]$ from party Y via oblivious transfer. Then, party Y retrieves element $\mathbf{v}_X[m_Y]$ from party X via oblivious transfer. Even though Y sends their value $\mathbf{v}_Y[m_X]$ first, as long as X is unaware of m_Y , and therefore whether the round is ultimately going to succeed or fail, X is unable to mount an attack against this value with any reasonable probability of success. A proof of this fact is provided in Theorem 4.8.9. In order to learn d_Y X must continue the protocol, and send their own value $\mathbf{v}_X[m_Y]$.

After receiving $\mathbf{v}_X[m_Y]$, Y can attempt to decrypt c_X .

$$Y : d_X \text{ or } \perp \leftarrow \mathcal{E}.\text{Dec}_{\mathbf{v}_X[m_Y]}(c_X) \quad (4.34)$$

If Y succeeds they can release b to X in good faith. X then uses b to decrypt the key k_Y , which they use to

decrypt the secret d_Y .

$$X : k_Y \leftarrow \mathcal{E}.\text{Dec}_b(\mathbf{v}_Y[m_X]) \quad (4.35)$$

$$X : d_Y \leftarrow \mathcal{E}.\text{Dec}_{k_Y}(c_Y) \quad (4.36)$$

Instead, if Y attempts to cheat, they confirm for X that they sent a brute-forceable ciphertext earlier in the protocol, $\mathbf{v}_Y[m_X] = c_{k_Y}$, containing the key to decrypt Y 's secret. With this information X can justify mounting a brute-force attack in reasonable time-frame t .

If $m_X \neq m_Y$ then neither party has learned any information relevant to decrypting the other party's secret, and so the round failed and both parties need to restart from key generation. Before restarting Y must prove the failure to X by revealing the value $\mathbf{v}_X[m_Y]$, which they could only obtain if $m_X \neq m_Y$.

A single round from the protocol is shown diagrammatically in Figure 4.6.

4.8.2 Proofs

Theorem 4.8.1 (Computational Completeness). For honest X and Y the probability of successfully exchanging secrets can be made arbitrarily close to 1.

$$\Pr[X(d_Y) \wedge Y(d_X)] \approx 1 \quad (4.37)$$

Proof. In a single round of this FE protocol, X and Y successfully exchange data if and only if $m_X = m_Y$. These values are the index in the vector $\mathbf{v}$ at which X and Y store information relevant to their secret, and it is also the element they access in the other party's vector $\mathbf{v}$.

As m is a randomly generated integer in the range $[0, M)$ the probability of exchanging data is in any single round of the protocol is:

$$\Pr[X(d_Y) \wedge Y(d_X)] = \frac{1}{M} \text{ for a single round} \quad (4.38)$$

By Assumption 4, M is a constant. It follows that the protocol can be repeated across as many rounds as is required for the probability of a successful data exchange to be arbitrarily close to 1.

$$\therefore \Pr[X(d_Y) \wedge Y(d_X)] \approx 1 \quad (4.39)$$

□

Computational Strong Fairness in the context of semi-honest adversaries means that for any possibly-incomplete honestly generated protocol transcript γ , if it is computationally intractable for X to extract d_Y it is also computationally intractable for Y to extract d_X , and vice versa.

In order to prove internal soundness within a single round the following lemmas prove protocol soundness after each respective message is sent. In each subsequent lemma the round transcript δ is extended with the next message as specified in Figure 4.6, and the adversary takes on the role of the recipient of the latest message. The first two lemmas are trivial, and reduce to an attack on the underlying encryption scheme used to construct the FE protocol. This is sufficient to prove the soundness of this FE protocol within a single round for semi-honest adversaries.

A Single Round of the Two Party Fair Exchange Protocol with Arbitrarily Sized Secrets

$X(\mathcal{E}, OT, k, k', d_X, M)$	$Y(\mathcal{E}, OT, k, k', d_Y, M)$	Transcript
$k_X \leftarrow \mathcal{E}.\text{KGen}(k)$ $c_X \leftarrow \mathcal{E}.\text{Enc}_{k_X}(d_X)$	$k_Y \leftarrow \mathcal{E}.\text{KGen}(k)$ $c_Y \leftarrow \mathcal{E}.\text{Enc}_{k_Y}(d_Y)$	
	$\xleftarrow{c_Y}$	$\delta_0 = \{c_Y\}$
	$\xrightarrow{c_X}$	$\delta_1 = \delta_0 \parallel \{c_X\}$
$m_X \leftarrow \$\{0, \dots, M-1\}$ for i in $\{0, \dots, M-1\}$ if $i = m_X$ $\mathbf{v}_X[i] \leftarrow k_X$ else $\mathbf{v}_X[i] \leftarrow \mathcal{E}.\text{KGen}(k)$	$b \leftarrow \mathcal{E}.\text{KGen}(k')$ $c_{k_Y} \leftarrow \mathcal{E}.\text{Enc}_b(k_Y)$ $z \leftarrow \$\{0, 1\}^k$ $m_Y \leftarrow \$\{0, \dots, M-1\}$ for i in $\{0, \dots, M-1\}$ if $i = m_Y$ $\mathbf{v}_Y[i] \leftarrow c_{k_Y}$ else $\mathbf{v}_Y[i] \leftarrow \mathcal{E}.\text{Enc}_b(z)$	
	$v_Y \xleftarrow{OT} \mathbf{v}_Y[m_X]$ $\mathbf{v}_X[m_Y] \xrightarrow{OT} v_X$	$\delta_2 = \delta_1 \parallel \{v_Y\}$ $\delta_3 = \delta_2 \parallel \{v_X\}$
	$o \leftarrow \mathcal{E}.\text{Dec}_{v_X}(c_X)$ if $o = \perp$ $r \leftarrow v_X$ else $r \leftarrow b$	
	$\xleftarrow{r}$	$\delta_4 = \delta_3 \parallel \{r\}$
if $r \in \mathbf{v}_X \wedge r \neq k_X$ return $\perp$ else $k_Y \leftarrow \mathcal{E}.\text{Dec}_r(v_Y)$ return $\mathcal{E}.\text{Dec}_{k_Y}(c_Y)$	return o	

Figure 4.6: Overview of a single round of the two party FE protocol supporting the exchange of arbitrarily sized secrets. The round transcript δ is specified at each stage of communication for use in later proofs.

PFairnessGame ₁ (δ)	
1 :	$d'_Q \leftarrow \mathcal{A}_P(\delta)$
2 :	if $d_Q = d'_Q$ then return 1
3 :	else return 0

Figure 4.7: For an FE protocol between arbitrarily labelled parties P and Q , with a set of possible protocol transcripts Δ , transcript $\delta \in \Delta$, and adversary acting as party P given by $\mathcal{A}_P$. If the adversary is able to extract the data d_Q from δ , they win.

Definition 4.8.1. For arbitrary party P define the advantage in the PFairnessGame defined in Figure 4.7 as

$$\text{Adv}_{\mathcal{A}_P}^{\text{fairness}}(\delta) = \Pr[\text{PFairnessGame}(\delta) = 1] \quad (4.40)$$

Lemma 4.8.2. Within a single round, when $\delta = \{c_Y\}$, for an adversary bound in time t

$$\text{Adv}_{\mathcal{A}_X}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.41)$$

Proof. Consider adversary $\mathcal{A}_X(\delta)$ capable of extracting d_Y from δ . This could easily be used to construct an adversary able to undermine the IND-CPA security of $\mathcal{E}(k)$ by forwarding ciphertexts to $\mathcal{A}_X$ for decryption.

$$\therefore \text{Adv}_{\mathcal{A}_X}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.42)$$

□

Lemma 4.8.3. Within a single round, when $\delta = \{c_Y, c_X\}$, for an adversary bounded in time t

$$\text{Adv}_{\mathcal{A}_Y}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.43)$$

Proof. Similarly, consider adversary $\mathcal{A}_Y(\delta)$ capable of extracting d_X from δ . This could easily be used to construct an adversary able to undermine the IND-CPA security of $\mathcal{E}(k)$ by forwarding ciphertexts to $\mathcal{A}_Y$ for decryption, where the first element of δ is an arbitrary ciphertext from $\mathcal{E}(k)$.

$$\therefore \text{Adv}_{\mathcal{A}_Y}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.44)$$

□

Lemma 4.8.4. Within a single round, when $\delta = \{c_Y, c_X, v_Y\}$, for an adversary bounded in time t

$$\text{Adv}_{\mathcal{A}_X}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.45)$$

Proof. There are two possible situations; the first is that $v_Y = c_{k_Y}$, an the key used to produce ciphertext c_Y encrypted under $\mathcal{E}(k')$; the other is that v_Y and c_Y are completely unrelated. The former case occurs with probability $\frac{\varepsilon}{M}$. By definition, $\mathcal{E}(k')$ is (t, ε) -secure. As such, an adversary has a $\frac{\varepsilon}{M}$ probability of success when blindly mounting an attack against v_Y . Otherwise the adversary must mount an attack against c_Y , which also has probability $\frac{\varepsilon}{M}$ of success.

$$\therefore \text{Adv}_{\mathcal{A}_X}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.46)$$

□

Lemma 4.8.5. Within a single round, when $\delta = \{c_Y, c_X, v_Y, v_X\}$ and it is computationally intractable for party X to extract d_Y from δ , for an adversary bounded in time t

$$\text{Adv}_{\mathcal{A}_Y}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.47)$$

Proof. As it is computationally intractable for party X to extract d_Y , it must be the case that $v_Y \neq c_{k_Y}$ and therefore $m_X \neq m_Y$. Because of this, v_X is completely unrelated to c_X and so is no help when decrypting it. Because of this, the adversary has no more information than they did in Lemma 4.8.3 and their advantage in the fairness game must be similarly bounded.

$$\therefore \text{Adv}_{\mathcal{A}_Y}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.48)$$

□

Theorem 4.8.6 (Single Round Computational Strong Fairness). In a single round of the two party FE protocol if, at any time, the ability of Y to extract d_X from transcript δ is computationally intractable, then for any probabilistic adversary $\mathcal{A}$ bounded in time t :

$$\text{Adv}_{\mathcal{A}_Y}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.49)$$

Similarly, if, at any time, the ability of X to extract d_Y from transcript δ is computationally intractable, then for any probabilistic adversary $\mathcal{A}$ bounded in time t :

$$\text{Adv}_{\mathcal{A}_X}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.50)$$

Proof. In the first case, when party Y is acting maliciously, the round transcript must be either $\delta = \{c_Y, c_X\}$ or $\delta = \{c_Y, c_X, v_Y, v_X\}$. The former case is proved in Lemma 4.8.3 and the latter is proved in Lemma 4.8.5. It follows that for all possible situations, if the ability of X to extract d_Y from δ is computationally intractable then

$$\text{Adv}_{\mathcal{A}_Y}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.51)$$

In the second case, when party X is acting maliciously, the round transcript must be either $\delta = \{c_Y\}$ or $\delta = \{c_Y, c_X, v_Y\}$. The former case is proved in Lemma 4.8.2 and the latter is proved in Lemma 4.8.4. It follows that for all possible situations, if the ability of Y to extract d_X from δ is computationally intractable then

$$\text{Adv}_{\mathcal{A}_X}^{\text{fairness}}(\delta) \leq \frac{\varepsilon}{M} \quad (4.52)$$

□

The protocol has been proven as sound as the underlying encryption scheme $\mathcal{E}(k)$, but only within single protocol round. It is not immediately obvious why party X cannot mount an attack where they gain many

RRXFairnessGame ₁ (k, k')	
1 :	$d_Y \leftarrow \$ D$
2 :	$d'_Y \leftarrow \mathcal{A}_X()^{\text{OTranscript}(d_Y)}$
3 :	if $d_Y = d'_Y$ then return 1
4 :	else return 0

Figure 4.8: The PFairnessGame from Figure 4.7 adapted to model repeat rounds of the protocol. The adversary acting as X has access to an oracle, $\text{OTranscript}(d_Y)$, which provides random FE protocol transcripts of length 3 (i.e. $\delta = \{c_Y, c_X, c_z\}$). d_Y is initially sampled from the set of all possible input data D , and is then used to instantiate the oracle.

partial single-round transcripts, then identify the weakly encrypted ciphertext containing Y 's encryption key and mount a relatively simple brute-force attack against $\mathcal{E}(k')$ to extract the it. This attack would see X engaging in the protocol and disconnecting after the transcript is $\delta = \{c_Y, c_X, v_Y\}$, and by repeating this process an expected M times, one of these samples should include $v_Y = c_{k_Y}$.

Unfortunately for X , if they cannot distinguish between encryptions of the key used by Y and random ciphertexts they must attempt the brute-force attack against each sample. Intuitively, given that $\mathcal{E}(k')$ is M times easier to attack than $\mathcal{E}(k)$, and X is expected to perform M brute-force attacks to extract d_Y , a Sybil-style attack is clearly not feasible as it would take as much effort as a direct attack against $\mathcal{E}(k)$.

The RRXFairnessGame, defined in Figure 4.8, provides a security game for this style of attack, and *Repeat Round Fairness* defines the advantage of an adversary in this game. If the number of calls to the transcript oracle, OTranscript , is bound by some value n , this variation is referred to as the “bounded RRXFairnessGame”. When the calls to the transcript oracle are bound as such, a transcript containing a relevant value of c_z is guaranteed to appear in the first n transcripts.

Definition 4.8.2 (Repeat Round Fairness). For party X define the advantage in the RRXFairnessGame defined in Figure 4.8 as

$$\text{Adv}_{\mathcal{A}_X}^{\text{rrfairness}}(k, k') = \Pr[\text{RRXFairnessGame}(k, k') = 1] \quad (4.53)$$

Similarly, bounded advantage is defined as

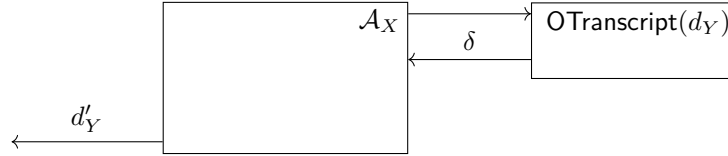
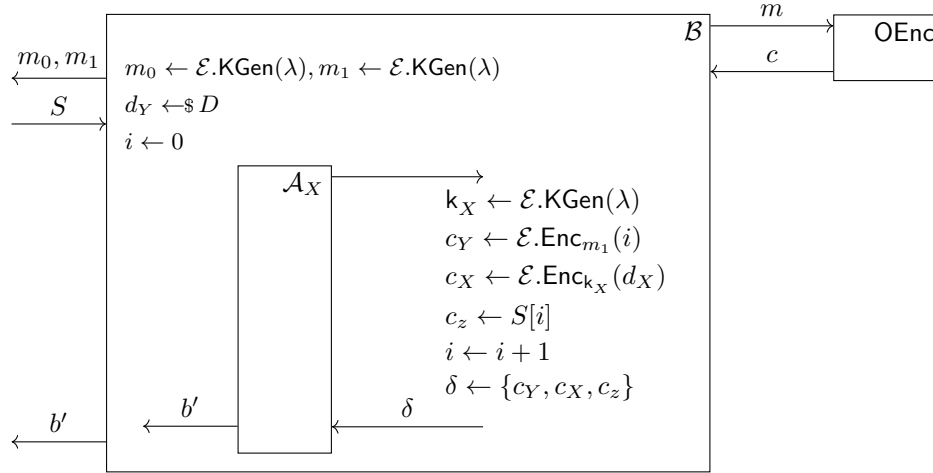
$$\text{Adv}_{\mathcal{A}_X}^{\text{brrfairness}}(k, k', n) = \Pr[\text{RRXFairnessGame}(k, k') = 1] \quad (4.54)$$

where the number of calls to OTranscript is bound by n .

The following sequence of lemmas, and ultimate proof, show that for values of M greater than a small lower bound, the ability to win the bounded RRXFairnessGame security game is no easier than winning the INDCPAGame against $\mathcal{E}(k)$. The sequence of proofs that follow show that:

- The n -bounded RRXFairnessGame(k, k') is at least as difficult as NINDCPAGame(k', n)
- There exists an $n_0 \leq M$ such that for all $n \geq n_0$ NINDCPAGame(k', n) is at least as difficult as INDCPAGame(k).

In conjunction these show that for M larger than some small lower bound the M -bounded RRXFairnessGame(k, k', M) is at least as hard as INDCPAGame(k).

Figure 4.9: Hypothetical adversary $\mathcal{A}_X$ for the bounded RRXFairnessGame .Figure 4.10: An adversary $\mathcal{B}$ capable of winning the NINDCPAGame given an adversary $\mathcal{A}$ capable of winning the n -bounded $\text{RepeatXFairnessGame}$. Whenever $\mathcal{A}$ requests a sample transcript the c_Y component is set to an encryption of the index i . When $\mathcal{A}$ is able to correlate the key used to encrypt the data they extract and return the index.

Lemma 4.8.7. The advantage of a probabilistic adversary bound in time t in the bounded RRXFairnessGame is less than advantage of a probabilistic adversary bound in time t in the NINDCPAGame with security parameter k' .

$$\text{Adv}_{\mathcal{A}}^{\text{brrfairness}}(k, k', n) \leq \text{Adv}_{\mathcal{B}}^{\text{nind-cpa}}(k', n) \quad (4.55)$$

Proof. This can be easily shown by a reduction from the bounded RRXFairnessGame to the NINDCPAGame . Consider an adversary, $\mathcal{A}$, capable of winning the RRXFairnessGame . The interface for adversary $\mathcal{A}$ is depicted in Figure 4.9.

Adversary $\mathcal{A}$ can be used to construct adversary $\mathcal{B}$ as given in Figure 4.10 which is able to win the NINDCPAGame . $\mathcal{A}$ is unable to extract a value for d_Y until it sees a transcript of the form $\delta = \{c_Y, c_X, c_z\}$ where $c_Y = \mathcal{E}(k). \text{Enc}_{m_1}(i)$ and $c_z = \mathcal{E}(k'). \text{Enc}(m_1)$ at which point it is able to extract i from c_Y . The other transcript samples provided to $\mathcal{A}$ by the oracle feature valid transcripts with uncorrelated values for c_Y and c_z .

$$\text{Adv}_{\mathcal{A}}^{\text{brrfairness}}(k, k', n) \leq \text{Adv}_{\mathcal{B}}^{\text{nind-cpa}}(k', n) \quad (4.56)$$

□

Lemma 4.8.8. For underlying encryption schemes $\mathcal{E}(k)$ and $\mathcal{E}(k')$ which are $(t, \frac{\epsilon}{M})$ -secure and (t, ϵ) -secure

respectively, given that $1 \leq \frac{M}{2} + \log_\varepsilon(M)$, there exists some $n_0 \leq M$ such that for all $n \geq n_0$

$$(\text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(k'))^{\frac{n}{2}} \leq \text{Adv}_{\mathcal{B}}^{\text{ind-cpa}}(k) \quad (4.57)$$

Proof.

$$(\text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(k'))^{\frac{n}{2}} \leq \text{Adv}_{\mathcal{B}}^{\text{ind-cpa}}(k) \quad (4.58)$$

$$\varepsilon^{\frac{n}{2}} \leq \frac{\varepsilon}{M} \quad (4.59)$$

Consider the smallest possible value for n for the inequality in Equation 4.59 to hold, n_0 :

$$\varepsilon^{\frac{n_0}{2}} = \frac{\varepsilon}{M} \quad (4.60)$$

$$\frac{n_0}{2} = \log_\varepsilon\left(\frac{\varepsilon}{M}\right) \quad (4.61)$$

$$n_0 = 2 \log_\varepsilon\left(\frac{\varepsilon}{M}\right) \quad (4.62)$$

$$= 2(\log_\varepsilon\left(\frac{1}{M}\right) + \log_\varepsilon(\varepsilon)) \quad (4.63)$$

$$= 2(\log_\varepsilon\left(\frac{1}{M}\right) + 1) \quad (4.64)$$

$$= 2(\log_\varepsilon(1) - \log_\varepsilon(M) + 1) \quad (4.65)$$

$$= 2(1 - \log_\varepsilon(M)) \quad (4.66)$$

It follows, for all $n \geq n_0$, where $n_0 = 2(1 - \log_\varepsilon(M))$

$$(\text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(k'))^{\frac{n}{2}} \leq \text{Adv}_{\mathcal{B}}^{\text{ind-cpa}}(k) \quad (4.67)$$

A bound for M can be generated such that there exists a valid $n_0 \leq M$ given the following sequence of equations.

$$n_0 \leq M \quad (4.68)$$

$$2(1 - \log_\varepsilon(M)) \leq M \quad (4.69)$$

$$1 \leq \frac{M}{2} + \log_\varepsilon(M) \quad (4.70)$$

Therefore, given $1 \leq \frac{M}{2} + \log_\varepsilon(M)$ there exists an $n_0 \leq M$ such that for all $n \geq n_0$

$$\begin{aligned} \therefore 1 &\leq \frac{M}{2} + \log_\varepsilon(M) \\ \implies \exists n_0 \leq M, \forall n > n_0, &(\text{Adv}_{\mathcal{A}}^{\text{ind-cpa}}(k'))^{\frac{n}{2}} \leq \text{Adv}_{\mathcal{B}}^{\text{ind-cpa}}(k) \end{aligned} \quad (4.71)$$

□

Theorem 4.8.9 (M -Round Strong Fairness). Given $(t, \frac{\varepsilon}{M})$ -secure encryption scheme $\mathcal{E}(k)$, (t, ε) -secure encryption scheme $\mathcal{E}(k')$, and $1 \leq \frac{M}{2} + \log_\varepsilon(M)$, the advantage of a probabilistic adversary bounded in t in

the M -bounded $\text{RRXFairnessGame}(k, k', M)$ security game, as defined in Definition 4.8.2, is no more than $\frac{\varepsilon}{M}$.

$$1 \leq \frac{M}{2} + \log_\varepsilon(M) \implies \text{Adv}_{\mathcal{A}}^{\text{brrfairness}}(k, k', M) \leq \frac{\varepsilon}{M} \quad (4.72)$$

Proof. By Lemma 4.8.7

$$\text{Adv}_{\mathcal{A}}^{\text{brrfairness}}(k, k', n) \leq \text{Adv}_{\mathcal{D}}^{\text{ind-cpa}}(k', n)$$

By Assumption 6

$$\text{Adv}_{\mathcal{D}}^{\text{ind-cpa}}(k', n) = (\text{Adv}_{\mathcal{E}}^{\text{ind-cpa}}(k'))^{\frac{n}{2}}$$

and so

$$\therefore \text{Adv}_{\mathcal{A}}^{\text{brrfairness}}(k, k', n) \leq (\text{Adv}_{\mathcal{E}}^{\text{ind-cpa}}(k'))^{\frac{n}{2}} \quad (4.73)$$

By Lemma 4.8.8

$$\begin{aligned} 1 &\leq \frac{M}{2} + \log_\varepsilon(M) \\ \implies \exists n_0 \leq M, \forall n > n_0, &\text{Adv}_{\mathcal{A}}^{\text{brrfairness}}(k, k', n) \leq \text{Adv}_{\mathcal{B}}^{\text{ind-cpa}}(k) \end{aligned} \quad (4.74)$$

Then, by setting $n = M$:

$$\therefore 1 \leq \frac{M}{2} + \log_\varepsilon(M) \implies \text{Adv}_{\mathcal{A}}^{\text{brrfairness}}(k, k', M) \leq \text{Adv}_{\mathcal{B}}^{\text{ind-cpa}}(k) \quad (4.75)$$

□

Note that the restriction on the number of observable rounds implied by Theorem 4.8.9 only restricts an adversarial X who repeatedly drops out of the protocol before revealing v_X and does not apply to an X who finished each round completely before embarking on the next. As such, the M bound on partial rounds does not conflict with the number of protocol rounds required to achieve *Completeness*.

Theorem 4.8.10 (Computational Strong Fairness). Given underlying encryptions schemes $\mathcal{E}(k)$ and $\mathcal{E}(k')$ which are $(t, \frac{\varepsilon}{M})$ -secure and (t, ε) -secure respectively, protocol transcripts that do not exceed M incomplete rounds, and $1 \leq \frac{M}{2} + \log_\varepsilon(M)$. If the ability of Y to extract d_X from any given multi-round protocol transcript is computationally intractable then an adversarial X bound in time t has probability no greater than $\frac{\varepsilon}{M}$ of extracting d_Y . Similarly, if the ability to extract d_Y from any given multi-round protocol transcript is computationally intractable then an adversarial Y bound in time t has probability no greater than $\frac{\varepsilon}{M}$ of extracting d_X .

Proof. The single round case is proved by Theorem 4.8.6, and Theorem 4.8.9 provides an adversarial bound for the multi-round case. □

4.8.3 Evaluation

This section has shown that the two party FE protocol for arbitrarily sized secrets against semi-honest adversaries has both the *Computational Completeness* and *Computational Strong Fairness* properties without the need for a trusted third party arbitrator even in the face of a multi-round attack. This result was previously believed to be impossible.

Two serious deficiencies of this scheme are that adversaries are assumed to be semi-honest and as k is made arbitrarily large with fixed k' , M must grow exponentially to reflect the growing difference between the difficulty of $\mathcal{E}(k)$ and $\mathcal{E}(k')$. The former is removed in the next section. With the latter the probability of a successful data transfer per execution of the protocol becomes vanishingly small. k is therefore restricted to smaller values that are deemed “secure enough”. This protocol, therefore, resists the security concept of *Forward Secrecy* [69]. In the FE setting this is not a problem, as both X and Y expect to eventually exchange secrets, but this could be an issue if the protocol never successfully terminates and one party begins an extended campaign of brute-force attacks against $\mathcal{E}(k)$.

The large failure rate of the scheme, $\frac{M-1}{M}$ is balanced by the fact M is a constant, and so the protocol is expected to be run a constant M times before a success. However, the multiple runs of the protocol cannot be parallelised, as this would falsify Lemma 4.8.4, and so they must be executed sequentially.

4.9 Two Party Fair Exchange with Malicious Adversaries

Removing the semi-honest adversary assumption required for the previous protocols necessitates adding cryptographic machinery to binds each participant to the message preconditions which were previously enforced by Assumption 8. As such, if either party were to deviate from the protocol at any stage, this can be detected and equated to the deviant party terminating early.

4.9.1 Additional Assumptions

Assumption 9. There exists an ZKP system with the properties of *Completeness*, *Soundness*, and *Zero Knowledge*.

Assumption 10. There exists an Commitment scheme which is both *Computationally Binding* and *Computationally Hiding*.

These additional cryptographic primitives are not tied to the concrete security parameters k and k' are instead instantiated with arbitrary security parameter λ .

4.9.2 Protocol Extensions

The first two messages in the protocol, when $\delta = \{c_Y, c_X\}$, require that both c_Y and c_X are valid encryptions of some data satisfying some pre-agreed functions Desc_X and Desc_Y respectively. To enforce this precondition both messages will be accompanied by a ZKP attesting to this constraint.

The next stage in the protocol involves party X learning element $\mathbf{v}_Y[m_X]$ from party Y , followed by Y learning element $\mathbf{v}_X[m_Y]$ from party X . The soundness of the protocol requires that this exchange satisfies the following properties:

- Neither party is influenced by the other’s choice of m , this property is already provided by the oblivious transfer backed exchange.
- The vector $\mathbf{v}_Y$ contains an element corresponding to a $\mathcal{E}(k')$ encryption of the key used to encrypt c_Y .
- Y send the actual selected value from $\mathbf{v}_Y$, not random noise or an otherwise dishonest substitution.

Interestingly, X does not need to be held to the same standards regarding $\mathbf{v}_X$ as Y does with $\mathbf{v}_Y$ for strong fairness to hold. That is because if X attempts to cheat and deny d_X to Y the protocol will never terminate and X will be able to do no better than mount the sampling attack proved computationally intractable in Theorem 4.8.9.

As such, the section of the protocol relating to the exchange of $\mathbf{v}_Y[m_X]$ for $\mathbf{v}_X[m_Y]$ is augmented to the following:

1. Y provides X with a vector of commitments, $\mathbf{c}_Y$ where each element is a commitment to it's corresponding entry in $\mathbf{v}_Y$.
2. Y provides X with a ZKPs attesting to the constraint that within $\mathbf{c}_Y$ at least one element corresponds to a commitment to a $\mathcal{E}(k')$ encryption of the key used to encrypt c_Y . This proof is then validated by X .
3. When X uses oblivious transfer to learn $\mathbf{v}_Y[m_X]$ that element is bundled with the randomness used for the commitment scheme, and the corresponding commitment, $\mathbf{c}_Y[m_X]$ is checked by X .

If at any point these constraints are violated, the participant equates the adversarial behaviour of their peer as an early termination. A single round in the extended protocol is given in Figure 4.11.

4.9.3 Properties

Theorem 4.9.1 (Computational Completeness). For honest X and Y the probability of successfully swapping their secrets is

$$\Pr[X(d_Y) \wedge Y(d_X)] \approx 1 \quad (4.76)$$

The proof for this is unchanged from Theorem 4.8.1 as the additional cryptographic machinery only enforces adherence to the protocol, which honest participants will satisfy regardless.

As an adversary can now deviate from the protocol, the single-round security game originally presented in Figure 4.7 is extended to allow the adversary to first present an incomplete and possibly partial round transcript δ which is then simulated to the point of an error by the challenger via **Simulate**, and returned to the adversary. **Simulate** fills in the elements of δ which would be left to the non-adversarial participant in the protocol up to the point of an honest participant returning an error. The adversary wins if they are able to extract the challenge data from the transcript while extracting a secret of their choosing from the transcript remains computationally intractable. This game is provided in Figure 4.13.

Definition 4.9.1. For arbitrary party P define the advantage in the **PExtFairnessGame**, defined in Figure 4.13, as

$$\text{Adv}_{\mathcal{A}}^{\text{extfairness}}(k, k', \lambda) = \Pr[\text{PExtFairnessGame}(k, k', \lambda) = 1] \quad (4.77)$$

Lemma 4.9.2 (Single Round Computational Strong Fairness). In a single round of the extended two party FE protocol, for any probabilistic adversary bound in time t , $\mathcal{A}$:

$$\text{Adv}_{\mathcal{A}}^{\text{extfairness}}(k, k', \lambda) \leq \frac{\varepsilon}{M} \quad (4.78)$$

Furthermore, if it is possible to extract both d_X and d_Y from transcript δ then d_X and d_Y satisfy Desc_Y and Desc_X respectively:

$$X(d_Y) \wedge Y(d_X) \implies \text{Desc}_X(d_Y) = \top \wedge \text{Desc}_Y(d_X) = \top \quad (4.79)$$

A Single Round of the Two Party Fair Exchange Protocol with Malicious Adversaries

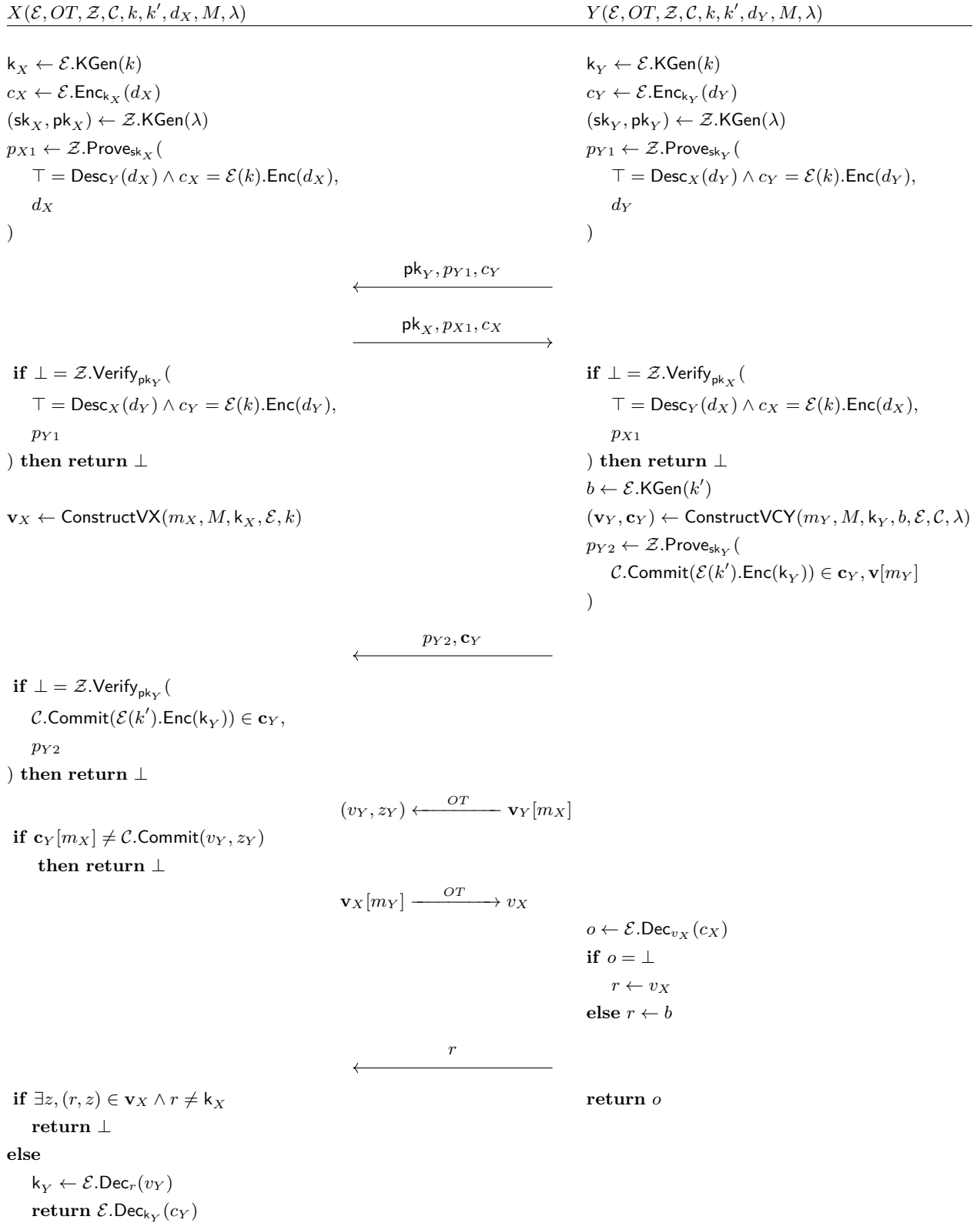


Figure 4.11: Overview of a single round of the two party FE protocol extended to be sound against malicious adversaries. Helper subroutines `ConstructVX` and `ConstructVCY` are defined in Figure 4.12.

ConstructVX($m, M, k, \mathcal{E}, k$)	ConstructVCY($m, M, k, k', \mathcal{E}, \mathcal{C}, \lambda$)
1: $\mathbf{v} \leftarrow [0; M]$	1: $\mathbf{v} \leftarrow [0; M]$
2: for i in $\{0, \dots, M\}$	2: $\mathbf{c} \leftarrow [0; M]$
3: if $i = m$	3: $u \leftarrow \$ \{0, 1\}^\lambda$
4: $\mathbf{v}[i] \leftarrow k$	4: for i in $\{0, \dots, M\}$
5: else	5: $z \leftarrow \$ \{0, 1\}^\lambda$
6: $\mathbf{v}[i] \leftarrow \mathcal{E}.\text{KGen}(k)$	6: if $i = m$
7: return $\mathbf{v}$	7: $\mathbf{v}[i] \leftarrow (\mathcal{E}.\text{Enc}_{k'}(k), z)$
	8: else
	9: $\mathbf{v}[i] \leftarrow (\mathcal{E}.\text{Enc}_{k'}(u), z)$
	10: $\mathbf{c}[i] \leftarrow \mathcal{C}.\text{Commit}(\mathbf{v}[i], z)$
	11: return $(\mathbf{v}, \mathbf{c})$

Figure 4.12: Helper subroutines for constructing $\mathbf{v}_X, \mathbf{v}_Y$, and $\mathbf{c}_Y$.

PExtFairnessGame ₁ (k, k', λ)
1: $d \leftarrow \$ D$
2: $\delta \leftarrow \mathcal{A}()$
3: $(\delta', d_{\mathcal{A}}) \leftarrow \text{Simulate}(\delta, d)$
4: $d' \leftarrow \mathcal{A}(\delta')$
5: if $d = d' \wedge d_{\mathcal{A}} = \perp$ then return 1
6: else return 0

Figure 4.13: the PExtFairnessGame security game. First random challenge data is sampled, d , then the adversary $\mathcal{A}$ is allowed to generate a partial transcript δ which only contains the messages that would be sent by one of the two participants. This transcript is then simulated by **Simulate**, which fills in δ with the corresponding messages of the other party until that party would terminate early given the protocol definition in Figure 4.11. **Simulate** returns $(\delta', d_{\mathcal{A}})$ where δ' is the transcript filled until an error was reached and $d_{\mathcal{A}}$ is the data “given up” by $\mathcal{A}$ during the simulation of δ .

Proof. As before, an adversarial X and Y will mount attacks at different stages of the protocol, depending on when they gain new information. This proof relies on the soundness against semi-honest adversaries proved in previous sections and bootstraps soundness against arbitrarily malicious adversaries. At each stage in the protocol it is shown that the protocol transcript must satisfy the preconditions previously enforced by Assumption 8, or a complexity assumption has been violated.

The transcript for a complete round is given by

$$\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X, p_{X1}, c_X, p_{Y2}, \mathbf{c}_Y, v_Y, z_Y, v_X, r\} \quad (4.80)$$

this can be derived from Figure 4.11.

Case $\delta = \{\mathbf{pk}_Y\}$: It is impossible for $\mathcal{A}$ to extract d_Y from the public key of an underlying proof system.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}\}$: Either the above is true, or $\mathcal{A}$ can be used to break the *Zero Knowledge* property on $\mathcal{Z}$. By Assumption 9 this is impossible.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y\}$: Either the above cases are true or $\mathcal{A}$ can be used to break the IND-CPA property on $\mathcal{E}(k)$. This has advantage $\frac{\epsilon}{M}$.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X\}$: It is impossible for $\mathcal{A}$ to extract d_X from a proving system public key.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X, p_{X1}\}$: Either the above cases are true or $\mathcal{A}$ can be used to break the *Zero Knowledge* property on $\mathcal{Z}$. By Assumption 9 this is impossible.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X, p_{X1}, c_X\}$: Either the above cases are true or $\mathcal{A}$ can be used to break the IND-CPA property on $\mathcal{E}(k)$. This has advantage $\frac{\epsilon}{M}$.

For transcripts any longer the protocol will terminate if the constraints expected of the proofs p_{Y1} and p_{X1} are unable to be verified. As such it can be assumed these have held. Importantly c_X and c_Y must each be encryptions of d_X and d_Y , each of which must satisfy Desc_Y and Desc_X respectively. By Assumption 9, an adversary attempting to circumvent the *Soundness* property on a proving system has advantage $\text{negl}(\lambda)$.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X, p_{X1}, c_X, p_{Y2}\}$: Either the above cases are true or $\mathcal{A}$ can be used to break both the *Zero Knowledge* property on $\mathcal{Z}$, which by Assumption 9 is impossible, and the *Computationally Hiding* property on $\mathcal{C}$, which by Assumption 10 has advantage $\text{negl}(\lambda)$.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X, p_{X1}, c_X, p_{Y2}, \mathbf{c}\}$: Either the above cases are true or $\mathcal{A}$ can be used to break the *Computationally Hiding* property on $\mathcal{C}$, which by Assumption 10 has advantage $\text{negl}(\lambda)$.

For longer transcripts the protocol will terminate if the constraints expected of the proof p_{Y2} are unable to be verified, and as such it can be assumed they've held. By Assumption 9, an adversary attempting to circumvent this has advantage $\text{negl}(\lambda)$.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X, p_{X1}, c_X, p_{Y2}, \mathbf{c}_Y, v_Y\}$: Either the above cases are true or $\mathcal{A}$ can circumvent the result provided in Lemma 4.8.4, which has advantage $\frac{\epsilon}{M}$.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X, p_{X1}, c_X, p_{Y2}, \mathbf{c}_Y, v_Y, z_Y\}$: Either the above cases are true or $\mathcal{A}$ can extract d_Y from randomness z_Y , which is impossible.

For longer transcripts the protocol will terminate if v_Y does not match a commitment from $\mathbf{c}_Y$, and as such it can be assumed this is the case. An adversary attempting to circumvent this will be violating the *Computationally Binding* property on $\mathcal{C}$, which by Assumption 10 has advantage $\text{negl}(\lambda)$.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X, p_{X1}, c_X, p_{Y2}, \mathbf{c}, v_Y, z_Y, v_X\}$: Either the above cases are true, or $\mathcal{A}$ can extract d_X from uncorrelated randomness v_X , which is impossible.

Case $\delta = \{\mathbf{pk}_Y, p_{Y1}, c_Y, \mathbf{pk}_X, p_{X1}, c_X, p_{Y2}, \mathbf{c}, v_Y, z_Y, v_X, r\}$: r is used to confirm the round was a failure

or generously assist X with extracting d_Y if Y has already extracted d_X . As such it can confer no advantage to an adversary, beyond that already identified in the above cases.

In the above cases the largest adversarial advantage is given by $\frac{\varepsilon}{M}$

$$\therefore \text{Adv}_{\mathcal{A}}^{\text{extfairness}}(k, k', \lambda) \leq \frac{\varepsilon}{M} \quad (4.81)$$

□

Lemma 4.9.2 only relates to the fairness within a round. However, as the fairness within the round has been enforced, Theorem 4.8.9 can be applied to derive fairness in the multi-round case.

Theorem 4.9.3 (Computational Strong Fairness). For underlying encryptions schemes $\mathcal{E}(k)$ and $\mathcal{E}(k')$ which are $(t, \frac{\varepsilon}{M})$ -secure and (t, ε) -secure respectively, for protocol transcripts that do not exceed M incomplete rounds, and $1 \leq \frac{M}{2} + \log_{\varepsilon}(M)$. If the ability of Y to extract d_X from any given multi-round protocol transcript is computationally intractable then an adversarial X bound in time t has probability no greater than $\frac{\varepsilon}{M}$ of extracting d_Y . Similarly, if the ability to extract d_Y from any given multi-round protocol transcript is computationally intractable then an adversarial Y bound in time t has probability no greater than $\frac{\varepsilon}{M}$ of extracting d_X .

Proof. Lemma 4.9.2 states that the ability of either X or Y to unfairly extract d_X from a single round transcript δ is

$$\text{Adv}_{\mathcal{A}}^{\text{fairness}}(k, k', \lambda) \leq \frac{\varepsilon}{M} \quad (4.82)$$

By Lemma 4.9.2 the participants are bound to act honestly within round, as to do otherwise would cause an early termination. Therefore, the result of Theorem 4.8.9 can be applied for the multi-round case:

$$\therefore \text{Adv}_{\mathcal{A}}^{\text{brffairness}}(k, k', M) \leq \frac{\varepsilon}{M} \quad (4.83)$$

□

4.9.4 Evaluation

With the additional cryptographic machinery applied to the protocol in this section fairness against arbitrarily malicious adversaries has been bootstrapped from fairness against semi-honest adversaries. This required the use of non-interactive ZKPs and commitment schemes to enforce message preconditions which were previously enforced by underlying assumptions about the capabilities of the adversary. This solved one of the fundamental flaws with the previous iteration of this scheme.

Unfortunately this scheme inherits both the high failure rate, and the “good enough” security of the ancestral semi-honest adversary protocol variant. The high failure rate flaw is compounded by the high computational cost of ZKPs which are now used each round to enforce message preconditions.

4.10 Asymmetry Between X and Y

There is an inherent asymmetry between X and Y in the protocols defined above. Taking the final protocol as an example:

- X must perform one expensive $\mathcal{Z}.\text{Prove}$ operation, whereas Y must perform two.
- Y can terminate before revealing r on any round and force X to expend effort to investigate the possibility they were the victim of cheating.

In earlier iterations of the protocol the asymmetry is even more pronounced as neither party was constructing ZKP proofs. It is important to note that although one of the two participants can be forced to expend a large amount of computational effort investigating the possibility they were the victim of cheating, this does not violate any FE requirements.

Due to this fact in any deployment of these protocols the actual assignment of X and Y to concrete participants should be randomised. This can be achieved between mutually distrusting parties by using commitment schemes to simulate a shared coin flip. Between concrete participants A and B :

1. A proposes that if the coin flip results in 0 then A will take on the role of X , and B will take on the role of Y . Otherwise A will take on the role of Y and B will take on the role of X .
2. B selects an arbitrary bit, $b_B \leftarrow \{0, 1\}$, and shares a commitment to this bit with A . $\mathcal{C}.\text{Commit}(b_B)$
3. A selects an arbitrary bit, $b_A \leftarrow \{0, 1\}$, and shares a commitment to this bit with B . $\mathcal{C}.\text{Commit}(b_A)$
4. Both parties then reveal their committed values. Both participants generate the shared coin flip by XORing the two values together $b_A \oplus b_B$, and take on their corresponding roles in the protocol.

Any unfairness in this shared coin flip undermines the *Binding* property of the chosen commitment scheme.

4.11 Applications to Consensus

Two party FE is inexorably linked to the consensus problem, as a two party FE scheme with the *Strong Fairness* property can be used to construct a two party consensus protocol [93]. A well known impossibility result shows that no deterministic asynchronous protocol can exist for the consensus problem [55]. As the final protocol constructed in this chapter is non-deterministic in lightly synchronous networks it cannot be used to construct a counter example for this proof. It can, however, be used as a black box in the construction linking FE and consensus to build a novel protocol for consensus. In this section a brief overview of the consensus problem is provided alongside a representative sample of algorithms for consensus. Then the construction relating two party FE to consensus in [93] is used to define a consensus protocol built from the two party FE scheme created in this work.

4.11.1 The Consensus Problem

The consensus problem is a foundational problem in computer science in which an honest majority of nodes in a network attempt to reach an agreement on some value [17]. Common applications involve agreeing on some swarm-wide action to take, applying transactions to a database, and deciding on an order for atomic network events, such as in an online video game. A consensus algorithm should have the *Termination*, *Agreement* and *Validity* properties, definitions for which are adapted from [97].

Definition 4.11.1 (Termination). The nodes eventually reach agreement.

Definition 4.11.2 (Agreement). Two non-faulty nodes cannot decide on different values.

Definition 4.11.3 (Validity). Only values from a predetermined set can be decided.

Computational systems can be modelled as either synchronous or asynchronous. The former assumes messages are sent and received in an order which is agreed upon network-wide. The impossibility result of [55] states that there can be no guarantee that an asynchronous consensus protocol will terminate. As such the *Termination* property is impossible in the asynchronous setting. In practice, a small possibility of failure is accepted.

4.11.2 Existing Consensus Schemes

Some of the most widely used consensus protocols are in the Paxos family of protocols [97]. Paxos requires that messages are not lost, reordered, or duplicated but does not require strong synchrony, which means the networks in this work need only be “partially synchronous”. There is a huge variety to the protocols in this family but the basic operation happens over a number of rounds which promote a suggested value v from a proposal, to a promise, and then finally to acceptance across a majority sub-network of non-faulty nodes. Each proposal for a value includes a separate integer x , greater than the x included with any previous proposed values. When the promise has been recognised by the sub-network of nodes they then reject any proposals with an x smaller than any proposals they have successfully promoted to the promise round.

More recent work in the fully asynchronous setting, Honey Badger [87], provides an atomic broadcast protocol making use of an asynchronous common subset primitive. The protocol operates with nodes suggesting transactions from a buffer of values, the union of these suggestions becomes the canonical set of transactions accepted by all nodes. This highly efficient scheme can apply tens of thousands of transactions a second in swarms of hundreds of nodes. Each communication round takes $\mathcal{O}(n)$ bandwidth for n nodes, and there are an expected $\Theta(\log n)$ rounds before consensus is reached.

Neither of these schemes contradict the impossibility result relating to the non-existence of asynchronous consensus protocols [55] as each has a insignificantly small probability of non-termination.

4.11.3 Constructing Consensus Protocols from Two Party Fair Exchange

The argument for the impossibility of two party FE is built around a construction in which an FE protocol with the *Strong Fairness* property is used to build a consensus protocol. This construction is detailed in Figure 4.14. Given a two party FE scheme this consensus construction makes up to four invocations of this scheme to generate agreement between the two participants.

4.11.4 Evaluation

Given the consensus protocol built on the final FE scheme presented in Section 4.9, each call to FE in the algorithm detailed in Figure 4.14 is expected to take M attempts to successfully terminate in an exchange of secrets. Given a random selection of v for both participants in the consensus protocol there is an even distribution of between one and four invocations to FE. As such there is an average of 2.5 calls to FE. This results in an expected $2.5M$ rounds of the FE scheme per consensus over $\{0, 1\}$. As the scheme can be used to reach consensus between two objects, represented as $\{0, 1\}$, it can be extended to differentiate between o objects by recursively composing this function to provide consensus in a binary-search style, where agreement between which of two progressively smaller groups of objects is decided. So ultimately deciding between o objects would require an expected $2.5M \log o$ invocations of FE. Performance would degrade dramatically

Consensus(v)	
1 :	$t \leftarrow \text{FE}(v, \text{Desc}(1))$
2 :	if $t \neq \perp$ then return 1 // If FE didn't fail, both parties agree $v = 1$
3 :	$t \leftarrow \text{FE}(v, \text{Desc}(0))$
4 :	if $t \neq \perp$ then return 0 // If FE didn't fail, both parties agree $v = 0$
5 :	if $v = 1$ then
6 :	$t \leftarrow \text{FE}(1, \text{Desc}(0))$
7 :	if $t \neq \perp$ then return 0 // Both parties have different values of v , so return 0
8 :	else return v // Unreachable with honest participants.
9 :	else
10 :	$t \leftarrow \text{FE}(0, \text{Desc}(1))$
11 :	if $t \neq \perp$ then return 0 // Both parties have different values of v , so return 0
12 :	else return v // Unreachable with honest participants.

Figure 4.14: Consensus protocol from two party FE as defined in [93] in which two nodes can reach consensus over a 1-bit value, $v \in \{0, 1\}$. The construction requires **Desc**, a function capable of producing a description of expected data, and **FE** which is a two party FE invocation that takes the sent secret, d_X , as the first argument and a description of the received secret, $\text{Desc}(d_Y)$, as the second argument. With honest participants this consensus construction returns 1 if both parties input 1, and 0 otherwise. When interacting with a dishonest peer a node will not change its mind from the original value v .

if the objects could not be partitioned into subsets that could be agreed upon in a binary-search-style. Extending this protocol to more than two nodes would involve invoking the consensus protocol between all nodes in the group pairwise. All nodes either maintain a decision of 1 or change their mind to a 0 if their peer has decided on a 0. Once each node has interacted with each other node they will all agree on a value, 1 if all of them selected 1, 0 otherwise. Any dishonest peers in the swarm will be unable to change the selected values of any of their peers. Optimising beyond this naive extension is outside the scope of this thesis, as is extending the protocol to more than two nodes.

The impossibility result relating to consensus schemes proves that any protocol that can support asynchronous consensus must have some probability of never terminating [55]. This proof is not contradicted by this work as the FE scheme presented in Section 3.5 has a $\frac{M-1}{M}$ failure rate on each individual FE attempt, and so there is a possibility that each successive attempt to exchange secrets results in failure.

Clearly, the consensus protocol constructed from FE is considerably worse than the state-of-the-art Honey Badger consensus protocol [87]. There would be substantial work required to extend the FE-consensus construction to support an arbitrary number of nodes, and the failure rate of the underlying FE scheme would no doubt produce a dramatic increase in the bandwidth required to reach a similar result as Honey Badger.

4.12 Further Work

There is clear value in two party FE, which for the first time, the constructions in this chapter show is possible. However, although the final protocol presented here is sound from a security perspective, it has some relatively serious performance concerns which would have drawbacks for any implementation. The high failure rate of $\frac{M-1}{M}$ requires the protocol to be run an expected M times. Depending on how cautiously M is selected and the security parameter of the “secure” encryption scheme k this can be very large and so the

bandwidth requirements can be very demanding. With the example values given to justify Assumption 4, that of $k = 84$ bits and $k' = 70$ bits, the expected number of rounds for the final FE protocol to successfully terminate is $\mathbb{E}[\text{number of rounds}] = 2^{14}$. This problem is only made worse through the use of expensive cryptographic primitives such as ZKPs.

The security of the scheme is forced to stay in the realm of “good enough” as to keep M relatively small. There is potential for further work in hardening the scheme in this regard. The relationship between the two instantiations of the encryption scheme $\mathcal{E}(k)$ and $\mathcal{E}(k')$ is what allows the failure rate to stay constant for a given situation even as adversarial abilities grow. But exploring this relationship further to achieve a lower M with better security guarantees would allow for the construction to move towards arbitrarily high levels of security.

This work has proved two party FE to be possible. Suddenly this opens up a whole new class of constructions as possible. Approaches to two party FE will be varied and this new class of solutions has yet to be explored.

4.13 Conclusion

To the best of the authors knowledge the constructions detailed in this chapter include the first FE protocol satisfying a notion of *Strong Fairness* not to require any trusted third party involvement. This result was previously assumed to be impossible, but in fact it is only impossible for a specific configuration of the problem, namely that of fully deterministic participants in an asynchronous environment. If arbitrarily small probability of failure is acceptable, a two party FE protocol is possible.

This work is used later in this thesis to support the exchange of digital goods between two otherwise anonymous nodes in a distributed and decentralised computation network. Two party FE is used to support the exchange of computational results for credits redeemable for computational effort. The use of FE in this way is explored further in the following chapter, which makes black box use of two party FE.

Chapter 5

DSCS: A Trusted Computation Mesh

This chapter assembles external existing work and the VC and FE works detailed in Chapters 3 and 4 respectively into a protocol to support secure-by-default execution of programs for a variety of security models. These protocols are a new class of cryptographic construction, dubbed a TCM, which provides a secure computational substrate from a swarm of disparate nodes of varying levels of trust. The protocol presented here relies on a number of existing works and the research progress made during the first half of this thesis.

The first objective is to develop the definitions of a secure distributed computation network, defined as a TCM, with well defined properties. Then, a protocol is developed called DSCS satisfying the properties of a TCM. A variety of application domains are identified where a TCM could be used to good effect, these domains have varying security requirements and provided in this chapter are a corresponding series of adversarial models. With incremental changes to DSCS the protocol satisfies the TCM properties in each adversarial model with corresponding moderate increases in protocol complexity. At time of writing this chapter and Chapter 6 are summarised in a yet-unpublished paper titled “DSCS: Towards An Extensible Secure Decentralised Distributed Computation Protocol”.

5.1 Introduction

Since the introduction of personal computation in the late 20th century the trend long observed has been the progressive migration from monolithic centralised systems towards a larger number of small specialised personal devices. Not long ago an entire family would have a single computer, and instead now the number of independent internet connected devices owned by each individual is growing steadily, with an estimated 71% of homes in North America containing at least one IoT device [78].

As stated in Chapter 1, the overarching goal of this thesis is to define a first-of-its-kind secure-by-default distributed computation protocol. Such a protocol would allow for distributed devices in a variety of settings to collaborate on the execution of programs. Through collaboration with other nearby devices the computational latency of individual task can be reduced by leveraging any spare computational capacity of their peers.

The domains targeted by this work are heterogeneous IoT distributions in the home, distributed edge devices in remote locations, and possibly even homogeneous devices looking for a self-organising load balancing system. Leveraging a wider network to increase the momentary computational ability of any individual par-

participant opens a whole new class of programs to these usually-computationally restricted devices. Suddenly it becomes possible for data to be collected, encrypted, and homomorphically processed without offloading computation to devices in some distant data centre. The use of FHE is massively advantageous for IoT devices and distributed sensors as without the use FHE device theft and adversarial data forensics could threaten data confidentiality. The current alternative for these distributed devices is to encrypt the harvested data with a public key encryption scheme which is decrypted as it reaches a computer less prone to tampering, such as a server in a secure data centre.

RNSs are a common acceleration technique for homomorphic evaluation, as popular FHE schemes tend to operate across polynomial rings with large integer coefficients. The RNS acceleration technique introduces a high degree of trivial parallelism to previously fully sequential programs, which makes it even more well suited to the parallel nature of distributed computation.

The capability to keep data encrypted and still process locally on edge devices would massively increase the utility of these edge devices, elevating them from mere sensors, while hardening them against data leaks through device theft and tampering.

5.1.1 Contributions

The contributions of this chapter are:

- Formally define Secure Multi User System (SMUS), a general class of abstract secure computation substrate.
- Formally define TCM, a subclass of SMUS which requires support for decentralised and distributed computation.
- Provide evaluation mechanisms for TCMs, including frameworks for security proofs.
- Identify possible classes of applications for TCMs, with corresponding adversarial behaviours, specifying three corresponding adversarial models: Nosy, Lazy, and Untrusted.
- Propose DSCS, a candidate TCM protocol, including a formal description of mechanisms, and proofs that it satisfies the properties of a TCM, with incremental changes to maintain the security properties in each of the adversarial models.

5.1.2 Assumptions

The work in this chapter is grounded in the assumption that homomorphic evaluation throughput will continue to improve and the number of small internet-connected devices will continue to grow. These trends will increase the benefits of using the TCM protocol defined in this work, DSCS, to distribute computation in otherwise insecure settings.

Proving TCM security properties for the constructions in this chapter requires the existence of the following cryptographic objects.

Assumption 11. An RNS-acceleratable IND-CPA secure LFHE.

Any scheme from the family of Ring Learning With Errors (RLWE) LFHE schemes would satisfy this assumption.

Assumption 12. VC with the *Soundness* property.

Assumption 13. Two party FE with some notion of *Strong Fairness*.

The first three of which are well studied fields, with a healthy variety of constructions with diverse underlying assumptions. Two party FE is the most speculative assumption, relying on work in this thesis. However, along with VC, two party FE is only required for the most advanced constructions presented in this chapter.

In order to simplify the protocol definitions in this chapter it is assumed that all inter-node communication is handled by a tamper-proof authenticated point-to-point message system, an assumption which in itself only requires the existence of EUF-CMA signature schemes. All network communication is assumed to be well ordered.

Assumption 14 (EUF-CMA Signature Schemes). There exists an Existential Unforgability under Adaptive Chosen Message Attack (EUF-CMA) secure signature scheme.

5.1.3 Performance Assumptions

Although not required for the cryptographic guarantees provided by the constructions within this chapter, the protocols presented were designed with certain node and network performance characteristics in mind.

- The participating nodes will have access to a sufficiently low latency interconnect such that it is not a limiting factor.
- The participating nodes will have access to a sufficiently high bandwidth interconnect such that it is not a limiting factor.
- The participating nodes will all have a broadly similar compute capacity.
- Nodes will become unreachable with negligible probability while participating in a collaborate computation.
- Requirements for light synchrony in the network are inherited from the black-box use of a two party FE scheme. Although two party FE does not inherently require network synchrony, the work of [93] rules out deterministic solutions in asynchronous networks, and without an asynchronous counter example assuming that some network synchrony is required is sensible.

5.2 Related Work

A lot of the background information relevant to this chapter is covered in Chapter 2, in particular Section 2.2 on distributed networks is particularly relevant. This section introduces the distributed computation related work with two existing examples.

One such example, BOINC [9] - a framework for distributed volunteer scientific computation, offers centralised distributed computation with few security guarantees. Networks built on BOINC rely on the trusted nature of a verified centralised organiser, who is able to delegate work to volunteer nodes at their discretion. Because worker nodes are volunteers and the node issuing work is centralised and verified there are few concerns about improper use of the network. Unfortunately this approach is inflexible, and discreet projects need to be organised and advertised so that volunteers are engaged and unified in their pursuit of a shared objective. In this way all collaborators are motivated to engage proactively in the operation of the network and little effort needs to be expended on hardening the network against motivated adversaries.

A second example of a well known distributed computation protocol is the blockchain backed smart contract, of which Ethereum [32] is the most well-known example. These approaches offer decentralised distributed computation with some improved security guarantees. Unfortunately, blockchain smart contracts are not a good solution for generic computation for a number of reasons. Firstly, programs are single threaded and can do nothing to take advantage of the number of nodes operating in parallel on the network, unless the author writes multiple smart contracts, and manually partitions the parallel portions of their code. Even then, the process by which smart contracts are executed causes a massive duplication of effort. Secondly, creating a single smart contract is prohibitively expensive for most purposes as the costs to deploy even single line application can reach many hundreds of dollars of equivalent cryptocurrency. Writing and debugging smart contracts is beyond the reach of most developers. The distributed nature of Smart Contracts is focused on decentralised verifiably correct execution in an untrusted environment, as such it does not prioritise efficient use of computation resource. The maintenance of a blockchain is inherently under-utilising the computational resources of the participants as they must spend computational effort to provide proof-of- x results, where x is some scarce resource, to the ledger. Another security consideration for blockchain-backed smart contracts is that any smart contract input data is written to the blockchain in perpetuity. This state of affairs has been improved in recent years with the advent of newer blockchain protocols, many of which utilise zero knowledge technologies to provide a degree of data anonymity [109].

Regardless, blockchain consensus protocols are backed by constraints on some limited resource, be it proof-of-work or some other proof-of- x . This fact by itself makes them ill-suited for supporting computation on a network of edge devices, which are often heavily resource constrained. Ultimately, blockchain-backed solutions are designed for a different set of objectives, none of which are maximising throughput and device utilisation.

5.3 Background

As in previous chapters, the general background given in Chapter 2 is relevant, but here further context is given for the work specific to this chapter.

5.3.1 Verifiable Computation

Unlike previous chapters referencing VC schemes, in this chapter VC schemes are used as a black box, the abstract interface to which is specified in the paragraphs to come. This abstraction aims to preserve the longevity of the protocols presented in this work in the face of a changing landscape of VC schemes.

As a reminder from Chapter 3, a VC scheme operates between a Verifier and a Prover. The former specifies the computational task, and the latter acts as a remote worker completing the specified task and providing results. The capability of the Prover to falsify the results to the task delegated to them by the Verifier should reduce to some known computationally intractable problem. The abstract functions that make up a VC scheme differ slightly from the function signatures provided in Chapter 3, as these earlier definitions are more restrictive and tailored to the constructions in that chapter. The more general functions provided here require the public key as input to the `Compute` function, which instead of producing the result r produces a proof p from which the correct result is extracted by `Vf`, if it is indeed correct. Here VC schemes, $\mathcal{V} = (\text{KGen}, \text{ProbGen}, \text{Compute}, \text{Vf})$, are defined as definitions to the following functions:

- **Key Generation:** $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(\lambda)$; generate a public and secret key pair.

- **Problem Generation:** $\sigma \leftarrow \text{ProbGen}_{\text{pk}}(f, x)$; encode arbitrary function f and input x into problem σ .
- **Compute:** $s \leftarrow \text{Compute}_{\text{pk}}(\sigma)$; evaluate problem σ , producing a proof s .
- **Verification:** $r \text{ or } \perp \leftarrow \text{Vf}_{\text{sk}}(\sigma, s)$; verify the proof s , given problem σ . Returning the output of $f(x)$ if it is accepted, and $\perp$ otherwise.

Definitions for *Completeness* and *Soundness* can be found in Subsection 3.2.2.

In order to support popular LFHE schemes, the protocol presented in this chapter is intended to operate natively on high order polynomial rings with large integer coefficients. This makes the VC protocol specified in Chapter 3 particularly well suited. With operands consisting of large integer coefficients the opportunity of a massive reduction in complexity for the Verifier over naive duplication of results is very high. Unfortunately, this would require the protocol forgoes any parallelism traditionally provided by a RNS. The Prover-optimal property of the VC scheme in Chapter 3 is particularly useful for this work as nodes are expected to switch between Prover and Verifier characteristics depending on the workload in question.

5.3.2 Two Party Fair Exchange

The FE problem, that of the equitable exchange of digital goods between two mutually distrusting parties is a natural problem in the design of secure protocols. It involves two parties, X and Y who wish to exchange data d_X for d_Y and d_Y for d_X respectively. Neither party wishes to expose themselves to the possibility of sending data without receiving the data of the other participant. There are a variety of existing FE schemes with notions of *Strong Fairness*, all of which require the intervention of a trusted third party. Before the work detailed in Chapter 4 the current state-of-the-art had “optimistic” trusted third party involvement, in which the trusted third party only needs to intervene in the case of a dispute between X and Y [79].

As proven in Chapter 4, it is possible to construct FE protocols without the involvement of a trusted third party. The FE scheme presented in that chapter has some weaknesses. As such, it is hoped that considerably more work can be done in the domain of two party FE to obtain protocols with better characteristics. As with VC, the interface into two-party FE is kept intentionally abstract in order to adopt any performance and security improvements in future two-party FE schemes. For similar reasons of protocol longevity as was used when deciding to abstract the interface to VC, two party FE is used as a black box. This abstracted interface is defined in [93], and consists of the following functions:

- **Description:** $e \leftarrow \text{Desc}(d)$; a function able to produce a description of a piece of data or expected piece of data d .
- **Fair Exchange:** $d_Y \text{ or } \perp \leftarrow \text{FE}(d_X, \text{Desc}(d_Y))$; a function called by both parties taking a piece of data and a description of a desired piece of data. It returns the desired data or $\perp$.

Definitions for the *Completeness* and *Strong Fairness* properties can be found in Subsection 4.2.2.

In the distributed computation protocol definitions that follow the exchange of digital goods via an FE scheme is given by the double-ended half arrow notation used in Figure 5.1, this reflects the bidirectional atomic transactions FE provides.

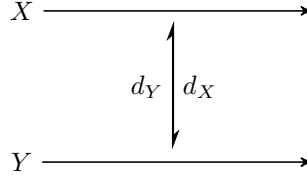


Figure 5.1: Nodes X and Y exchange digital goods via an FE protocol. Node X sends d_X and receives d_Y and node Y sends d_Y and receives d_X .

5.3.3 Homomorphic Encryption

FHE provides the ability to compute over data while it is encrypted. By removing the requirement that data is decrypted before being processed, encrypted private data can be sent to untrusted remote servers without fear that the remote servers will be able to learn any private information. This technology has applications in the secure processing of a wide variety of data: processing medical data in the remote cloud, processing data harvested from IoT sensors in a private company's servers, protecting financial transaction processing servers from the possibility of data exfiltration from server compromises.

More technically, a homomorphic cryptosystem is any encryption scheme which allows for predictable operations to be applied to data obfuscated within a ciphertext without decrypting the ciphertext. FHE schemes allow for both addition and multiplication operations. The earliest FHE schemes [59] are far from practical to implement but are important for their novelty. Modern schemes, such as CKKS [37] and BFV [52], have greatly improved execution speeds, and can provide encryption at rest and in-flight for sensitive data with a large linear impact on execution time.

The constituent elements of a FHE are defined in the following functions:

- **Key Generation:** $(pk, sk) \leftarrow \text{KGen}(\lambda)$; a key generation function producing a public key and corresponding secret key.
- **Encryption:** $c \leftarrow \text{Enc}_{pk}(m)$; an encryption function.
- **Decryption:** $m \leftarrow \text{Dec}_{sk}(c)$; a decryption function.
- **Addition:** $c_3 \leftarrow \text{EvalAdd}(c_1, c_2)$; an addition function in which for all $c_1 \leftarrow \text{Enc}_{pk}(m_1)$ and $c_2 \leftarrow \text{Enc}_{pk}(m_2)$, c_3 is a valid encryption of $m_1 + m_2$.
- **Multiplication:** $c_3 \leftarrow \text{EvalMul}(c_1, c_2)$; a multiplication function in which for all $c_1 \leftarrow \text{Enc}_{pk}(m_1)$ and $c_2 \leftarrow \text{Enc}_{pk}(m_2)$, c_3 is a valid encryption of $m_1 m_2$.

The key generation, encryption, and decryption functions are consistent with the definitions for a public key cryptosystem. However, FHE includes two additional functions corresponding to ciphertext addition and multiplication. This additional functionality maps well onto the arithmetic circuit representation of general computation, as previously discussed in Chapter 2.

In this work, an LFHE is chosen over a FHE to improve throughput at the cost of runtime flexibility. An LFHE offers a variation on FHE that has much lower latency for performing operations over ciphertexts with the additional requirement that the number of computational steps must be known at key generation. This is ideal for the system presented in this work as the computational depth can often be calculated in advance of executing a program, and where it cannot be calculated in advance cryptosystem parameters can be generated dynamically during execution to support specific sequences of computation.

BFV [52] is an example of one such LFHE and is the cryptosystem chosen to drive the homomorphic operations in the DSCS implementation in Chapter 6. BFV has seen widespread implementation due to an execution speed that approaches practicality and relative implementation simplicity. The implementation details of BFV are omitted as they are unimportant for the contributions of this chapter, instead, as in other cryptographic constructions presented in this thesis, the interface to a generic LFHE scheme is defined for use in this work.

RNSs were presented earlier in this thesis as a common acceleration technique for arithmetic over large integers. This concept is extended to a vector space by the Double Chinese Remainder Transform (DCRT) [85] which allows for its use accelerating polynomial arithmetic. NTT, RNSs, and DCRT are used interchangeably in the related work to refer to an extension to common RNS over integers to the domain of polynomials. BFV is a lattice cryptosystem, and so is a valid candidate for this acceleration technique. DCRT introduces a similar degree of parallelism as the RNS mechanism it is based on. This parallelism is leveraged to drive LFHE evaluation across a distributed network of collaborators. As such, the black box LFHE scheme used in the constructions within this chapter must be compatible with this acceleration technique.

Table 5.1 compares features of contemporary cryptosystems with homomorphic properties. Alongside BFV, another popular implementation choice is CKKS [37], which offers a LFHE scheme with can operate on floating point numbers and yields approximate results. Approximation error is configurable when the cryptosystem parameters are selected.

5.3.4 Signature Schemes

Signature schemes, $\mathcal{S} = (\text{KGen}, \text{Sign}, \text{Vf})$, provide a mechanism for verifying the identity of a message sender. They are specified by definitions for a key generation function, a signature function, and a verification function. A message is signed by the Sender and a Verifier can use the verification function in conjunction with a public key to verify the identity of the Sender. Implicitly, any communication between nodes in DSCS is sufficiently authenticated using a contemporary signature scheme, but signature schemes are also used as a building block when constructing a tit-for-tat credit system. As with other cryptographic building blocks used throughout this thesis, the implementation details are unimportant, and as such the precise scheme used does not matter. References to signature schemes are therefore kept abstract.

A signature scheme is defined by the following functions:

- **Key Generation:** $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(\lambda)$; a key generation function producing a public and private key pair.
- **Signing:** $s \leftarrow \text{Sign}_{\text{sk}}(m)$; a signing function used to generate signature s on message m given secret key sk .
- **Verification:** $\top \text{ or } \perp \leftarrow \text{Vf}_{\text{pk}}(m, s)$; a verification function that returns $\top$ if s is a valid signature on m and the provided public key pk corresponds to the private key used to generate signature s . Returns $\perp$ otherwise.

Later in this chapter the EUF-CMA security property of signature schemes is used to reason about the security properties of derived constructions. Informally this common security property provides the strong guarantee that given access to an oracle capable signing arbitrary messages, an adversary is unable to create a forged signature for a message without first sending the message to the signing oracle [106].

More formally, the definition of a EUF-CMA secure signature scheme is:

Name	Homomorphic	Implementation Ready	Library	DCRT Parallelisable	Notes
RSA [99]	Partially	Yes			
NRTU [72]	Partially	Yes		Yes	
Gentry [59]	FHE	No			
LTV [81]	LFHE	Yes		Yes	Recently discovered to be insecure [6]
BFV [52]	LFHE	Yes	SEAL	Yes	Popular choice for integer arithmetic
CKKS [37]	LFHE (approximate)	Yes	HEAAN & SEAL		Popular choice for floating point arithmetic

Table 5.1: Comparison of some cryptosystems with homomorphic properties

EUFCMAGame ₁ (λ)	
1 :	(pk, sk) ← KGen(λ)
2 :	(m, s) ← $\mathcal{A}^{\text{OSign}_{\text{sk}}}(\text{pk})$
3 :	r ← Vf _{pk} (m, s)
4 :	if r = ⊤ then return 1
5 :	else return 0

Figure 5.2: The EUF-CMA security game. (1) a key pair, (pk, sk) is generated, (2) and the adversary $\mathcal{A}$ must provide a message m and signature s such that s is a valid signature on m under sk. The adversary is provided with the public key pk and an oracle capable of generating signatures under sk on arbitrary messages OSign_{sk} . (3,4) the adversary wins if they can provide a valid signature and message pair where the message has not been sent to the signing oracle.

Definition 5.3.1 (EUF-CMA Secure Signature Scheme). The advantage of adversary $\mathcal{A}$ in the EUF-CMA security game, detailed in Figure 5.2 as:

$$\text{Adv}_{\mathcal{A}}^{\text{euf-cma}}(\lambda) = \Pr[\text{EUFCMAGame}(\lambda) = 1] \quad (5.1)$$

A signature scheme is EUF-CMA secure if for any probabilistic polynomial time adversary $\mathcal{A}$:

$$\text{Adv}_{\mathcal{A}}^{\text{euf-cma}}(\lambda) \leq \text{negl}(\lambda) \quad (5.2)$$

5.3.5 Threat Model

When discussing any application with security implications it is important to carefully consider the threat model and clearly define adversarial capability. The threat models listed here become incrementally stronger as more abilities are added to an adversary. Broadly, the work in this chapter operates in the following three threat models, in order of escalating adversarial capability: *Nosy Nodes*, *Lazy Nodes*, and *Untrusted Nodes*. Protocol security is maintained by incrementally extending the protocol as adversarial capability escalates.

The lowest threat model is that of a network of mostly trusted devices.

Definition 5.3.2 (Threat Model: Nosy Nodes). No participating nodes will deviate from the protocol and all nodes will engage enthusiastically with the network. Nodes will not maliciously submit tasks to the network, and will only attempt to evaluate tasks on the network that they would evaluate locally otherwise. However, the confidentiality of private data must be maintained in the face of a motivated adversary.

This is reflective of a closed network of homogeneous devices, such as a data centre deployment of identical servers, or uniform distributed sensors in a secure location. Nodes are guaranteed to act in the administrators best interests, with no incentive to deviate in any way from the tenants of the protocols in place.

The next threat model characterises nodes who do not deviate from the prescribed protocols but can engage lazily with the work of their peers.

Definition 5.3.3 (Threat Model: Lazy Nodes). No nodes will act maliciously but some may act lazily. They will not deviate from the network protocols, but they may delay responses in order to reduce the amount of computational work they do. The confidentiality of private data must be maintained in the face of a motivated adversary.

Nodes will act in their own self interest and will not offer up their resources freely. This is more indicative of the relationship between nodes in a non-homogeneous IoT deployment. Devices from competing vendors are not incentivised to donate their computational resources to each other as to do so would improve the latency of operations for a competing brand.

Finally, the strongest threat model involves subjecting the network to adversaries that can act arbitrarily maliciously to disrupt network activity.

Definition 5.3.4 (Threat Model: Untrusted Nodes). A minority of the nodes in the network may act arbitrarily maliciously.

Adversaries are never a majority of the nodes in the network. This is the adversarial situation for an open network in which nodes can join freely and anonymously.

5.4 Secure Multi-User Systems

A sensible first step in constructing a secure decentralised and distributed computation protocol is to define the expected properties of general secure computation systems. In this chapter definitions for a class of systems named SMUSs are provided formally. Informally an SMUS is the class of general multi-user computation systems with strict separation of user-space data and program isolation. SMUSs are a broad family of computational platforms, from multi-user operating systems, to cloud-based Functions as a Service (FaaS). The important qualities that connect these disparate systems into a unified family are proper protections for the users of the systems from other users of the same system. Unprivileged users sharing a computer should not, by default, be able to tamper with the data or running processes of other users.

Practically these systems tend to share a unified storage medium (be it a single hard drive or a hierarchy of data centre storage solutions), as well as a unified processing medium (be it a single processor or a fleet of rack-mounted computers). The use of these resources is carefully mediated by well defined administrative processes such that the restrictions placed on unprivileged users are maintained.

The capability of users to bypass these restrictions by side channels or exploitation is considered outside the scope of this thesis, as these momentary breaches in the intended behaviour are often patched out of existence. Adversarial behaviours are considered later in this chapter, but within the more restricted context of a TCM. It is assumed that privileged users and services operate as intended to regulate the access of unprivileged users.

An SMUS therefore is a computational substrate that carefully regulates the resource access of unprivileged users and maintains well defined boundaries between user processes and data. The behaviours of these systems satisfy three minimal concepts. The first of these is *Data Availability*, this relates to the ability of users to freely access and manipulate their own data and launch programs that can do this on their behalf. The second is *Data Confidentiality*, this restricts the ability of users to arbitrarily read data belonging to other users. The third is *Data Integrity*, this property restricts the ability of users to arbitrarily overwrite data belonging to other users. More formally, an SMUS is any system for computation comprised of the following components:

- A set of identities, I , which are associated bijectively with users of the system.
- A key-value data store, $V : K \rightarrow D$, mapping keys K to data D , with the following interface:
 - **Read:** $d \leftarrow \text{Read}(k)$, read data d with key k .

- **Write:** $\text{Write}(k, d)$, write data d with associated key k .
- Access control logic, $A \subseteq (I \times K)$.

These components are composed into a system for general computation as detailed in the definition for *Multi-User Computation Platform*.

Definition 5.4.1 (Multi-User Computation Platform). Users with identities $i \in I$, read and write data $(k, d) \in V$, and launch programs in order to process their data.

In addition to being a functional multi-user computation platform, an SMUS guarantees certain security properties relating to the separation of user data. The definitions for *Data Availability*, *Data Confidentiality*, and *Data Integrity* follow in the remainder of this section.

Definition 5.4.2 (Data Availability). If $(i, k) \in A$ then user with identity i can successfully call $V.\text{Read}(k)$ and $V.\text{Write}(k, d)$.

The *Data Availability* property has the emergent behaviour that users must be able to launch programs $p \in P$ that can manipulate data in accordance with the systems access control policy. The access control model encapsulated by the *Data Availability* property is simpler than that available to most modern multi-user operating systems. These tend to have fine-grained access control logic supporting the separation of read and write access to data. This fine-grained approach can be adapted to this model by restricting the permissioning mechanism to grant both read and write access to data in a single operation. As such the more fine-grained approach to access control is a subset of the model presented by A .

The *Data Availability* property ensures the system has the expected behaviour for well behaving users of a system for computation, in which users can read and write to data locations to which they have been granted explicit access. In order to secure the data of users, two behaviours must be explicitly blocked. The first is the ability of a user to read data held at a position that they do not hold permission to access. The second is the ability of a user to write data to a location they do not have permission to access. These map to the *Data Confidentiality* and *Data Integrity* properties respectively.

Definition 5.4.3 (Data Confidentiality). An identity cannot access data in any way that would violate the access control logic.

Definition 5.4.4 (Data Integrity). An identity cannot write data to the key-value data store V if it would violate the access control logic.

As with *Data Availability*, where the definition has the added implication that user-launched programs have the ability to read and write data, *Data Integrity* requires that users cannot manipulate the processes running on behalf of other users. If users were able to affect the running processes of their peers, they would gain the ability to tamper with data as it is in-flight, and therefore contradict the *Data Integrity* property.

Together, these properties define an SMUS.

Definition 5.4.5 (Secure Multi User Systems (SMUS)). An SMUS is a *Multi-User Computation Platform* that has the *Data Availability*, *Data Confidentiality* and *Data Integrity* properties.

SMUSs form a diverse solution space, one example of which is the modern multi-user operating system. It has users I , who run programs, on a computer with a data store of data keyed by memory locations V . The operating system itself runs administrative processes with elevated privileges, which ensure all data

access satisfies access control parameters A and handles cleanly context switching between isolated running applications. As previously mentioned, the access control mechanisms tend to be more fine-grained than the minimum required capabilities of an SMUS.

Another widespread SMUS is a general FaaS application deployment. Here, a cloud service provider allocates server compute resources to form a computational substrate on which functions, bundled into a FaaS software packages by account holders I , are executed. The cloud provider satisfies a similar role to the multi-user operating system administrative services in the previous example and acts as a authoritative overseer ensuring both *Data Confidentiality* and *Data Integrity* conditions are satisfied.

Blockchain smart contracts, such as those offered by the Ethereum cryptocurrency [32], also satisfy the SMUS definition, this time without the use of a central authority, albeit with some caveats. Network participants, I , operating nodes on the blockchain run a replicated state machine to execute smart contracts in exchange for cryptocurrency tokens. The caveat is that when a user writes a smart contract and there is the implicit assumption that any required input data is publicly viewable and therefore must grant read permissions to the entire network, $A = I \times D$, this trivial case alleviates all access control issues. There is some notion that zero knowledge technologies can be used to remove this caveat and provide fine-grained access controls and user data privacy [109]. However, the computational throughput of smart contracts are ill-suited to the computationally demanding task of FHE processing. Similarly, prohibitively high costs reduce the viability further.

There is a further requirement for SMUSs, implicit in their definition as computation platforms - this is the requirement that the SMUS is relatively performant and as such offer a compelling medium for computation. After the properties guaranteeing system security, as outlined above, are satisfied the metric by which most SMUSs are measured is the speed with which they deliver correct results to proposed computational programs. A standard abstraction of the computational throughput of a system is the number of operations per second it is capable of, and as such in this work the informal notion of system performance is reflected as operations per second.

5.5 Trusted Computation Mesh

With the expected behaviours of an SMUS codified it's definition, consider instead an SMUS that must also be distributed and decentralised. In these hypothetical systems swarms of nodes will operate as a computational substrate, collaborating on tasks securely. This setting presents a host of new issues and security concerns relating to administrating data access, protecting the correctness of running programs, and the unfair use of computation resources across users of the network. This section extends the definition of an SMUS with additional well-defined restrictions to create a sub-class of solutions: the class of TCMs. Beyond the definitions inherited from the class of SMUSs, a TCM must be distributed, decentralised, fairly allocate computational resources, and have a low overhead for participating in the platform. The requirement to have a low overhead when running the protocol lowers the barrier to entry, a necessity for many of the potential areas of application, which often feature resource constrained edge devices in remote locations.

Consider a set of edge devices deployed in a field, each of which collects data and has an intermittent computational task to perform. Individually, they run tasks when conditions allow, relying on regular breaks in computation when conditions are ill-suited, such as during the night for solar-powered devices. In such a domain, a TCM will provide the ability for these devices to collaborate, self-organise, and organically load-balance computational tasks across the entire network. Through this collaboration the latency of com-

putational tasks is reduced, and devices better saturate their computational capability. This ability also results in a reduction in the need for devices to store energy in order to continuously work on their own individual tasks. This is ideal for devices with severe constraints on power consumption, and may not have the capability to store energy for any reasonable amount of time. Similarly, devices may be constrained by their ability to dissipate heat, and collaboration of the sort allowed by a TCM would expand this constraint to the thermal mass of the entire network. By networking these devices together they can support the computational tasks of their peers while maximising their utility and computation throughput, all while the network minimises the latency of their own tasks.

The first additional constraint on the system involves ensuring the platform is distributed across a network of nodes. The system is distributed if identities I are bijectively associated with computation resources C . These pairings are called *Nodes*. Due to the nature of a network collaborating on computational tasks, discreet resources must be made available to all identities on the network. In practice, a single physical device can support multiple identities, but importantly each cannot fully saturate the physical device and some internal mediation must be used to partition the computational resource. This is abstracted to modelling the entire network as nodes $n \in N$ which contain an identity $i \in I$ and a computation resource $c \in C$, and the existence of a bijective function $f : I \rightarrow C$.

Definition 5.5.1 (Distributed SMUS). In addition to satisfying the properties of an SMUS, the identities of users I and the set of compute resources C are paired into a set of nodes $N \subseteq C \times I$. All computation resource is associated with an identity by a bijective function $f : C \rightarrow I$ such that $f(c) = i$ if $(c, i) \in N$.

Due to the bijective relationship between identities and computation resource, if the *Distributed SMUS* property holds, references to I and C are equivalent to N .

Definition 5.5.2 (Decentralised SMUS). In addition to satisfying the properties of an SMUS the system contains no processes with elevated privileges.

The *Decentralised SMUS* property removes the ability for an administrative identity to mediate the behaviour of unprivileged users. This property requires that the rules of system itself must support the rigorous enforcement of the access control policy, and other administrative roles such as the proper separation of user processes.

There is the additional problem in a distributed and decentralised setting of the unfair allocation of computational resources. Without a centralised authority mediating the distribution of network computation resources there is no guarantee that devices connecting to the network will gain value from participation. To ensure a device is treated fairly in the network, the protocol itself should support the fair distribution of computational resources in the network over time. As such, over time the resources allocated to the tasks of any node should be proportional to the computational power brought to the network by the individual device.

Definition 5.5.3 (Fair Resource Allocation). For a node (i, c_i) the networked computational resources allocated to their computational tasks, c'_1 , is at least proportional to their own local computational resource c_1

$$c'_1 \propto c_1 \tag{5.3}$$

The Fair Resource Allocation property adds the guarantee that asymptotically, a malicious user can waste

no more computational power on frivolous programs than they would be able to alone, disconnected from the network.

Currently, all the above definitions can be satisfied by some blockchain-backed smart contracts platform, albeit with a very low computational throughput. However, any trust built from blockchain requires considerable resources to operate. In the most common consensus mechanism, proof-of-work, the operational computational overhead would dwarf the utility provided by performing useful operations. Devices would be continuously performing the proof-of-work operations rather than providing useful computational utility. The *Low Longevity Cost* property is introduced to add a sensible upper bound to the demands of network participation.

Definition 5.5.4 (Low Longevity Cost). For network lifetime t , the minimum operational storage requirement, o_d , of participating in the network is:

$$o_d \leq \text{negl}(t) \quad (5.4)$$

If local storage requirements are negligible in the runtime of the network then the barrier to entry for edge devices is reduced. Otherwise, making significant contributions to the network would only be possible for non-resource constrained devices. Many of the devices that would profit from a TCM are those with heavy restrictions on resources and they would be unable to participate. Due to this requirement there can be no mandated network history, and therefore most blockchain solutions are explicitly disqualified.

Definition 5.5.5 (Trusted Computation Mesh (TCM)). A TCM is an SMUS (as per Definition 5.4.5) which satisfies the additional requirements of a *Distributed SMUS* (Definition 5.5.1), *Decentralised SMUS* (Definition 5.5.2), as well as maintaining *Fair Resource Allocation* (Definition 5.5.3) with *Low Longevity Cost* (Definition 5.5.4).

In order to build a TCM some collaboration mechanism must be designed which facilitates the shared execution of programs between nodes on the network, without relying on a centralised authority or on blockchain technology. This requires developing a new mechanism by which the swarm of nodes can self-organise and collaborate while resisting the influence of malicious actors. The network should grow in an ad hoc manner to support new nodes, without the need for any single point of organisation.

Figure 5.3 highlights the relationship between the TCM and SMUS definitions as a space of possible solutions. TCMs are a more restricted SMUS, with added restrictions to carve out an interesting and unexplored region in the solution space. Namely, the fact that the secure computational substrate is a decentralised swarm of distributed nodes. As a subset of an SMUS, TCMs inherit the security restrictions of an SMUS, and as such are suitable for general purpose multi-user computation without fear of adversarial tampering or exploitation by peers.

5.6 DSCS in the Nosy Node Threat Model

This section details DSCS, a candidate TCM. It is a protocol designed to support a decentralised network of independent non-homogeneous nodes collaborating on computation. The data confidentiality approach chosen involves flagging sensitive data and using black box RNS-acceleratable LFHE in order to preserve data integrity.

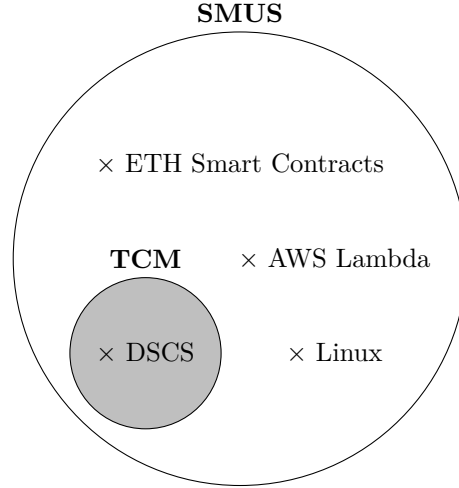


Figure 5.3: SMUS solution space of secure multi-user computation platforms, with the TCM subspace highlighted.

The secure computation protocol presented in this chapter is specified first in the *Nosy Node* security model (Definition 5.3.2), which is representative of a deployment within a closed network of mostly-trusted nodes. This is followed by proofs supporting the claim that DSCS is a TCM in this setting. Then, incremental additions are made to the protocol to maintain the TCM properties in security models with a more capable adversary. The additions to DSCS support TCM properties in the *Lazy Node* model (Definition 5.3.3) and *Untrusted Node* model (Definition 5.3.4) in turn.

As the DSCS protocol runs, the distributed nodes dynamically organise into smaller sub-networks to work on specific tasks. The sub-networks are not mutually exclusive and nodes can belong to as many sub-networks as they wish. For completeness some new terminology is used here: programs executed in the protocol are referred to as *Work*; *Work* is broken into atomic chunks with finite well-defined execution time, these are referred to as *Work Fragments*. When in the *Nosy Nodes* adversarial model DSCS operates in its most basic variation.

A step-by-step overview of the operation of the protocol follows in the next subsection. After the overview individual workflows are shown in more detail. These flows are then combined to provide a state machine articulating the public interface nodes present to their peers on the network. A second state machine is included that provides some the internal mechanisms required by later reasoning in regard to properties of the scheme. Unlike private behaviours the public interface must be clearly defined with all nodes conforming to expected patterns of behaviour.

In this distributed computation protocol Nodes behaviours are partitioned into *Work-Issuing Nodes* and *Worker Nodes*, the former coordinates the execution of a specific program and the latter assists in the execution of the programs of their peers. Nodes often fluctuate between these two classes of behaviour, and can often be both at the same time - organising the execution of their own programs while simultaneously assisting peers with the execution of other programs. A common theme throughout the rest of this work is that the expected behaviours of a *Work-Issuing Node* are relatively well defined, but the behaviours of *Worker Nodes* are not enforced by the protocol. This is due to the fact that the public interfaces nodes present to their peers, which must be well defined, are concerned with their responsibilities as a *Work-Issuing Node*. By leaving *Worker Node* behaviours undefined, the arguments about system security are stronger, as

there are no assumptions regarding conformity to the protocol.

5.6.1 Overview

The objective of DSCS is to support the acceleration of computation via the organisation of networked nodes into a computational substrate allowing collaboration to reduce the latency of executing programs. Initially, nodes join the network by making contact with any existing network node. The existing DSCS node shares their knowledge of the network topology with the connecting node. Practically, this is achieved via an off-the-shelf mDNS [38] protocol which operates in parallel to the DSCS protocol as a standard solution for continuous network discovery. The network topology and reasoning about the organisation of the distributed network is beyond the scope of this thesis.

Node-defined programs are converted into *Work Fragments* which are designed to capture a small part of a program with a finite execution time. The *Work Fragments* are designed to have a 1-to-1 correspondence with an arithmetic circuit. If the number of iterations of each loop and the direction of each branching instruction is known at compile-time the conversion from programs to *Work Fragments* can be done at compile-time. Alternatively, for programs with dynamic control flow dependent on variables only known at run-time, the conversion to *Work Fragments* can be done dynamically during the execution of the program as the results of completed *Work Fragments* are processed. Raw programs are never seen on the network, and the conversion from programs to *Work Fragments* is left to the implementer. Due to the fact *Work Fragments* must have a finite execution time they require a restricted instruction set to avoid any run-time ambiguity. Programs should therefore be defined in a more verbose programming language and then converted client-side into *Work Fragments*, a high level workflow for which is diagrammed in Figure 5.4. Because *Work Fragments* are used network-wide to define parts of shared computational tasks their composition and content must be standardised.

The first DSCS variant presented in this chapter targets the most constrained security model, that of the *Nosy Node*, and therefore is the most simple DSCS variant presented in this chapter. Nodes passively advertise programs for which they are the *Work-Issuing Node*, and if a node wishes to contribute to the ongoing computational effort they simply communicate this with the *Work-Issuing Node* who then adds the new *Worker Node* to an internal registry. A node may decide to not advertise work for any number of reasons, but one such is that they feel the already registered *Worker Nodes* have saturated the parallelism of the tasks and so any additional nodes would do no more than incur additional operational overhead.

Worker Nodes in a *Work-Issuing Node*'s registry are repeatedly sent *Work Fragments* until they become unreachable or express disinterest in any more work. This continues until the task is completed. *Worker Nodes* send the results of completed *Work Fragments* to the *Work-Issuing Node* and the *Work-Issuing Node* updates their internal state accordingly. If any *Worker Nodes* are on the registry for a task but there are no *Work Fragments* ready to be worked on, they must wait for dependencies in pending *Work Fragments* to be resolved.

Work Fragments include some data and one or more operations to apply to that data. When work is dispatched to a node there should be no external dependencies. Some of the data in a *Work Fragment* is flagged as the output of the work fragment. When the work is completed the data flagged as output returned to the organising node. Operations can be chained together with no prescriptive limitations beyond fundamental architectural constraints imposed by the implementation libraries and system hardware. The reduced instruction set supported by *Work Fragments* ensures the expected computational cost is easily predicted in advance. The instruction set includes basic arithmetic operations and a simple fixed loop instruction to

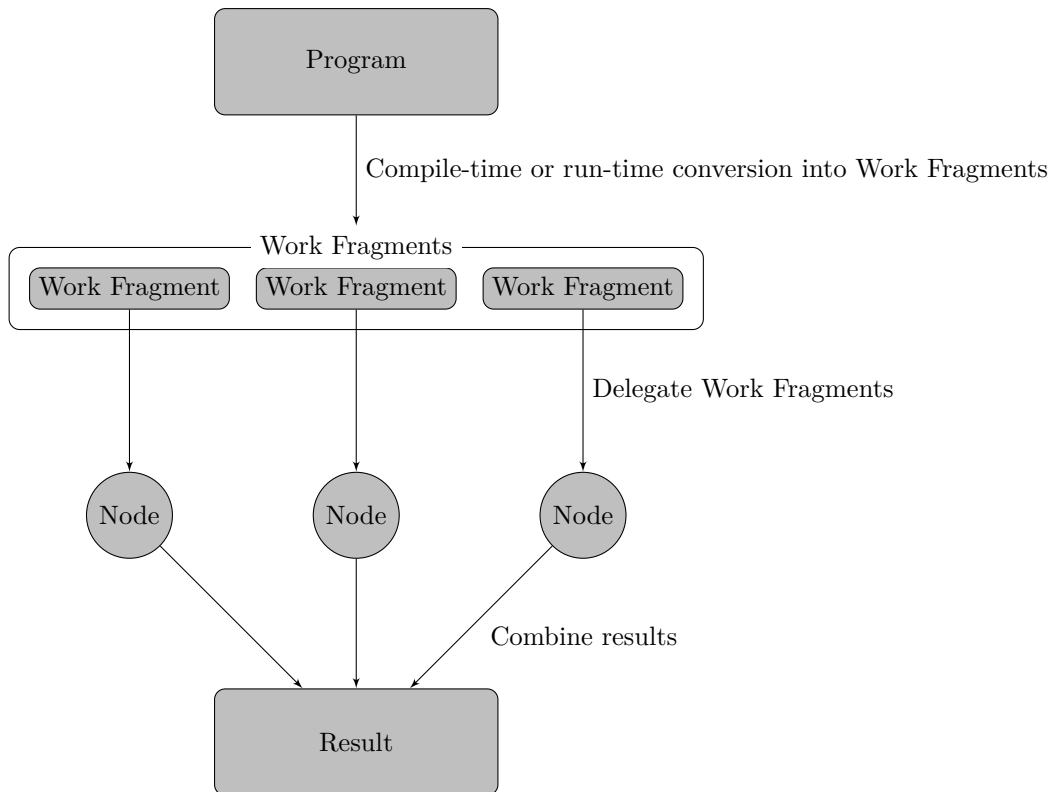


Figure 5.4: The high-level workflow for programs executed on DSCS. Nodes derive *Work Fragments* from Programs at either compile-time or run-time, these are then delegated to nodes in the network, who produce results that are combined into an overall result.

compress repeated instructions. Reducing the available instructions to only support simple data-types and execution times easily calculable before run-time is a common approach used in privacy-preserving delegated computation [95]. As with cryptographic approaches that use similarly reduced instruction sets, this work inherits the ability to trivially represent *Work Fragments* as arithmetic circuits. This will be a useful feature when this protocol is later extended to support the verification of results via a VC scheme.

As a privacy-preserving method DSCS supports interleaving plaintext operations with LFHE operations. In order to support LFHE the instantiation of the cryptosystem occurs as the *Work Fragments* are generated. This allows the parameters to be tuned to precisely support the computational depth of the *Work Fragment*. The operands in a *Work Fragment* are integers, but they can be represented in plaintext, as an LFHE encrypted ciphertext, or as a single RNS-encoded LFHE ciphertext fragment. Operations support mixed argument types and automatic conversions are made where necessary.

If a programming language is able to express which variables are to remain private and which are allowed to be publicly viewable this can be reflected in the *Work Fragment* as operands can be flagged as private. At run time any *Work Fragment* due to be worked on with operands flagged as private must be evaluated locally on the *Work-Issuing Node* or its operands must be converted into an encrypted LFHE ciphertext form before being dispatched to a *Worker Node*. This decision can be made dynamically depending on any number of factors such as the computational throughput of the network or the available resources of the *Work-Issuing Node*.

5.6.2 Preliminary Definitions

The notion of referring to a node and its behaviours relative to a specific executing program was introduced earlier as *Work-Issuing Nodes* and *Worker Nodes*. A more formal definition for both is provided here, alongside a new definition, the *Work Group* which provides a collective noun for a set of *Worker Nodes* collaborating on a specific task.

Definition 5.6.1 (Work-Issuing Node). The *Work-Issuing Node* is the node organising the task currently being referenced.

Definition 5.6.2 (Work Group). The *Work-Issuing Node* maintains a registry of nodes involved in the current task called a *Work Group*.

Definition 5.6.3 (Worker Node). A *Worker Node* is any node in the work group.

Of course, a node can be both a *Work-Issuing Node* and a *Worker Node* at the same time depending on which in-flight computation the discussion is relative to. Conveniently, most adversarial behaviours are constrained to *Worker Node* behaviours.

The behaviours required by nodes participating in the DSCS protocol are shown as a pair of interacting state machines. In order to reflect the capabilities of any adversarial nodes few restrictions are placed on the behaviour of *Worker Nodes* beyond those prescribed by the security model. As such, the protocol can be mostly defined by a single state machine defining *Work-Issuing Node* behaviours, from which transitions are triggered by a combination of shifting internal logic and external communication from other nodes on the network. All the public-facing interfaces into nodes are encapsulated by their *Work-Issuing Node* behaviours and as such a well-defined interface can be created. This approach also allows the mechanisms around *Worker Nodes* behaviours to be left to individual implementation details. Of course, in this work it is assumed that nodes do not deviate from the given security model, but this model becomes much less restrictive in subsequent sections.

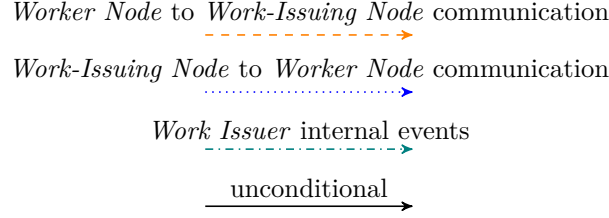


Figure 5.5: A key for the state transitions for DSCS. State transitions colour coded to match whether the state transition was prompted by a *Worker Node* to *Work Issuing Node* communication (orange), *Work-Issuing Node* to *Worker Node* communication (blue), internal *Work-Issuing Node* logic (green) or an unconditional transition (black).

In the diagrams to follow labelled states and colour coded state transitions are used to show expected behaviour work-flows. A key to the colour coded state transition labels is provided in Figure 5.5. A DSCS *Work-Issuing Node* can be in one of many possible states at any given time, for any of the target security models the *Work-Issuing Node* can be in state:

- *N*: Start state, neutral.
- *I*: Pending **INCOMPLETE** *Work Fragments*.
- *P*: Pending **READY** *Work Fragments*.
- *Q*: Processing advertising work request, r_a .
- *J*: Processing joining work group request, r_j .
- *S*: Pending **READY** *Work Fragments* and idle *Workers* in *Worker Group*.
- *C*: Processing results of delegated *Work Fragments*, r_c .
- *R*: Processing leaving work group request, r_r .

In later security models the following additional state is used

- *D*: Processing credit redemption request, r_w .

Transitions are often activated by internal events. These events often relate to the state of any pending *Work Fragments* and any registered *Worker Nodes*. These two sets are denoted by F and W respectively, and boolean expressions relating to their composition can often be seen on the label for a state transition. If this condition evaluates to true, then the transition is taken. The set of *Work Fragments* in the **WAITING** state is given by F_W , the set of *Work Fragments* in the **READY** state is given by F_R , and the set of *Work Fragments* in the **COMPLETE** state is given by F_C . Similarly, the set of registered *Worker Nodes* currently with a delegated task is given by W_T and the set of registered *Worker Nodes* without a delegated task is given by W_I .

As messages are assumed to arrive at their intended destination unaltered, they are queued by the recipient who processes them in order. The node changes state by popping a message off the queue and processing it, or by responding to some internal event. For the sake of diagrammatic simplicity state machines presented in the following pages are non-exhaustive, and some uninteresting transitions are omitted. Because of this simplification a naive implementation of the following protocols would result in a system vulnerable to a trivial denial of service attack. This attack involves a malicious node simply sending messages to their

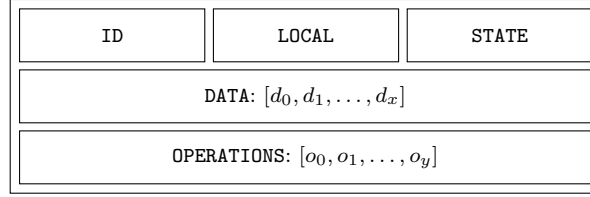


Figure 5.6: DSCS Work Fragment.

peers that do not correspond to transitions from their current state. One such attack could involve sending message r_j (a request to join a *Work Group*) to a node in state N (no pending *Work Fragments*). A protocol implementation can become resistant to this exploit by ensuring nodes prioritise internal events over the external queued messages, and dropping external messages if they do not have an associated transition from the current state.

Work Fragments are a data structure describing a self-contained program with a finite execution time. These structures contain an integer **ID**, a boolean flag **LOCAL** used to distinguish *Work Fragments* containing plaintext private data, a **STATE** indicating the current stage the *Work Fragment* is in, data **DATA** and some **OPERATIONS** to perform on that data. This structure is diagrammed in Figure 5.6. The **STATE** can be set to **WAITING**, **READY**, or **COMPLETE** depending on the state of any external data dependencies and if it has been executed or not.

A **WAITING** *Work Fragment* is stalled until an external data dependency is resolved, a **READY** *Work Fragment* has no unresolved external dependencies but has not yet been executed, a **COMPLETE** *Work Fragment* has been executed. When a *Worker Node* receives a *Work Fragment* it should have no unresolved external dependencies, and as such be in the **READY** state. Some of the **DATA** is flagged as output data and when a *Worker Node* finishes executing a *Work Fragment* they return the set of all **DATA** entries marked as output. Typical example *Work Fragments* are given in Figure 5.7.

Supported data types in *Work Fragments* include:

- **INT**: Plaintext integers.
- **CT**: LFHE encrypted ciphertexts.
- **CTF**: Ciphertext fragments, a single LFHE encrypted ciphertext residue in RNS representation.

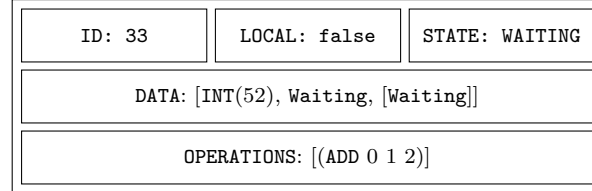
Plaintext and ciphertext representations coexist in order to allow mixed sensitivity data to be efficiently operated on, as well as allowing dynamic conversion at runtime.

Supported arithmetic operations in *Work Fragments* include:

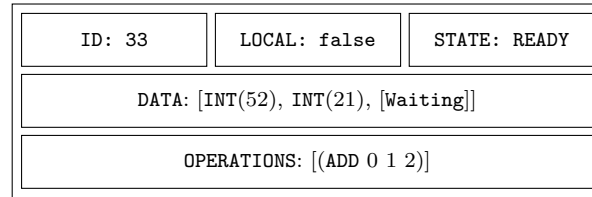
- **ADD** $d_1 \ d_2 \ r$: addition over integers, ciphertexts, and/or ciphertext fragments. $\text{DATA}[r] = \text{DATA}[d_1] + \text{DATA}[d_2]$
- **SUB** $d_1 \ d_2 \ r$: subtract integers. $\text{DATA}[r] = \text{DATA}[d_1] - \text{DATA}[d_2]$
- **MUL** $d_1 \ d_2 \ r$: multiply integers, ciphertexts, and/or ciphertext fragments. $\text{DATA}[r] = \text{DATA}[d_1] \times \text{DATA}[d_2]$
- **DIV** $d_1 \ d_2 \ r$: divide integers. $\text{DATA}[r] = \text{DATA}[d_1] / \text{DATA}[d_2]$

These instructions were chosen as a simple yet expressive subset of a more conventional instruction set. Importantly each instruction has a finite execution time that is known in advance.

Control meta operations allowed in *Work Fragments* include:



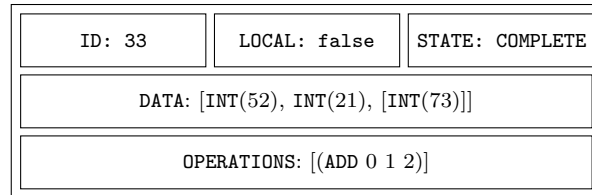
(a) A typical **WAITING** DSCS fragment. `DATA[1]` is currently waiting for an external dependency to be resolved. `DATA[2]` is where the final result of the *Work Fragment* will be written.



(b) A typical **READY** DSCS fragment. All external dependencies have been resolved. It is ready to be sent to a remote *Worker Node*.



(c) The data returned by the remote *Worker Node*.



(d) A typical **COMPLETE** DSCS fragment, updated with the result as provided by the remote *Worker Node*.

Figure 5.7: Typical DSCS *Work Fragments* in each of the three *Work Fragment* states, and the results object returned by a remote *Worker Node*. This fragment has been approved for remote execution. The single element of `OPERATIONS` is an `ADD` and takes as input the 0th and 1st elements of `DATA`. The operation is popped, executed, writes the output to the second element of `DATA`. The second element of `DATA` is flagged as the output of this *Work Fragment*, here denoted by an extra set of square brackets.

- **LOOP** $i\ o$: execute a loop a fixed number of times. i is the number of iterations, o is the instruction index to jump to.

Traditional branching operations such as **IF** and **ELSE** have been intentionally omitted in order to guarantee a fixed and easily predictable execution time for *Work Fragments*. If *Work Fragments* did not have an easily predictable execution time, then in more relaxed security models greedy malicious actors could claim they did more work than they actually did, and refuting this would require knowledge of how each of the branching instructions resolved.

Work Fragment instructions are interpreted in-sequence and change the entries in the set **DATA**. The *Work Fragment* can be trivially converted into an arithmetic circuit by replacing sequences of instructions with chained arithmetic gates and unrolling any **LOOP** instructions. Similarly, arithmetic circuits can be trivially converted into *Work Fragments* by replacing wires with a shared **DATA** pool and replacing arithmetic gates with DSCS instructions. Conceptually, conditional branching instructions can be included, if the security model assumes adversaries will not exploit the ambiguity introduced in *Work Fragment* execution time. VC could be used to validate either claims made about the results of conditional branches. However, in order to provide a simple and unified interface into DSCS only instructions which do not introduce any ambiguity are allowed. The introduction of conditional branches is left to future work.

The only supported non-arithmetic operation in a *Work Fragment* is the fixed **LOOP**. These are included as an easily-unrollable mechanism for compressing repeated operations. No control flow operations are allowed in *Work Fragments* to maintain the guarantee of a finite easily compile-time calculable execution time.

A concrete example of a target program being converted into *Work Fragments* is provided in the following chapter, in Subsection 6.3.3. This is left until then due to a dependency on the internal machinery of a DSCS node, unimportant to the public-facing interface and expected behaviours of DSCS nodes.

5.6.3 Protocol Definition

In the following state transition diagrams some states are circled in red, this indicates a *reactive* state, in which the *Work-Issuing Node* is idling. This is in contrast to non-reactive states in which the *Work-Issuing Node* takes some immediate action and then moves to an appropriate next state.

Starting a program



Figure 5.8: *Work-Issuing Node* transitions from state **Neutral** to state **Pending Incomplete Work Fragments** as a result of an pending *Work Fragments* not in the **COMPLETE** state. The same node then transitions to **Pending READY Work Fragments** as a result of an internal event which indicates there are pending *Work Fragments* in the **READY** state.

Each node begins in state **Neutral**, and when a task is internally queued for DSCS network execution the node transitions to the **Pending Incomplete Work Fragments**. The internal event triggering this transition indicates there are *Work Fragments* that are not in the **COMPLETE** state, meaning they are waiting to be delegated to a *Worker Node*. The node then transitions to state **Pending READY Work Fragments** as a result of *Work Fragments* in the **READY** state. These transitions are diagrammed in Figure 5.8. For diagrammatic simplicity,

most interactions implicitly force the *Work-Issuing Node* to return to state N , from which events can fire again to transition it back to the true reactive state.

Advertising Work

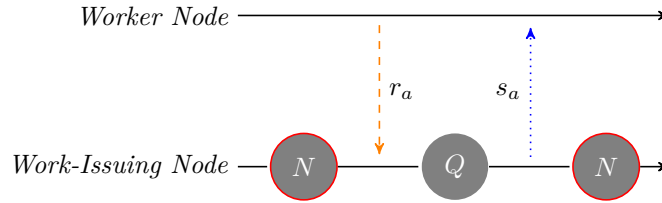


Figure 5.9: *Work-Issuing Node* receives a request from a *Worker Node* interested to know if there is any pending work, r_a . The *Work-Issuing Node* replies with s_a , which in this case is NO.

At any time a node can receive a request r_a from a *Worker Node* interested to learn if this *Work-Issuing Node* has any *Work Fragments* not in the **COMPLETE** state, this is shown in Figure 5.9. The *Work-Organising Node* responds with s_a which is either NO, or YES depending on whether there are any *Work Fragments* pending execution. The *Work-Issuing Node* may respond with NO even if there are *Work Fragments* that are yet to be evaluated, potentially because the size of their existing *Work Group* saturates the parallelism of the task.

Joining a Work Group

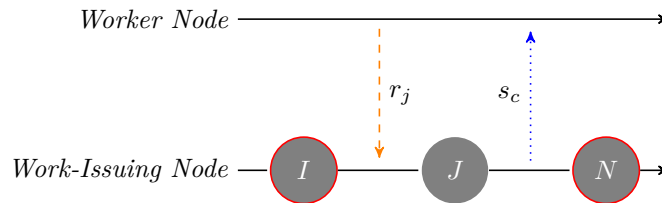


Figure 5.10: A *Work-Issuing Node* receives a request, r_j from a *Worker Node* to join an existing *Work Group*. The *Work-Issuing Node* sends a confirmation s_c , and transitions to state Neutral.

A *Worker Node* can join an existing *Work Group* by sending an appropriate request to the organising *Work-Issuing Node*, a process showing in Figure 5.10. After the *Work-Issuing Node* receives the request they add the originating *Worker Node* to the *Work Group* and send a confirmation to the *Worker Node*. The *Worker Node* should then expect to receive *Work Fragments* in the **READY** state as they become available.

Issuing Work to a Worker Node

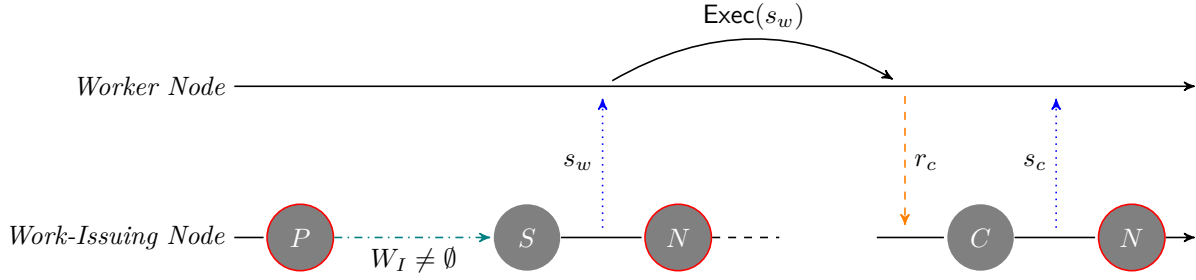


Figure 5.11: The *Work-Issuing Node* transitions from the **Pending READY** Work Fragments state to **S** when there are *Worker Nodes* in the *Work Group* without an assigned *Work Fragment*. The *Work Fragment* is sent to the *Worker Node* in message s_w to be evaluated by function Exec . The array of result data, r_c , is sent from the *Worker Node* to the *Work-Issuing Node* who processes the output and sends confirmation s_c .

When a *Work-Issuing Node* has *Work Fragments* in the **READY** state, and there are one or more available *Worker Nodes* in their *Work Group* the *Work-Issuing Node* sends *Work Fragment* s_w to the first available node. This node evaluates this fragment, and returns vector of results r_c to the originating *Work-Issuing Node*. This node then processes the result, updating any dependencies in subsequent *Work Fragments*, possibly dynamically generating new *Work Fragments*, and sends a confirmation to the *Worker Node*. The *Worker Node* is then added to the back of the queue of idling *Worker Nodes*.

Leaving a Work Group

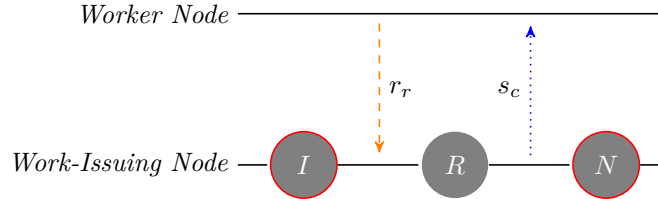
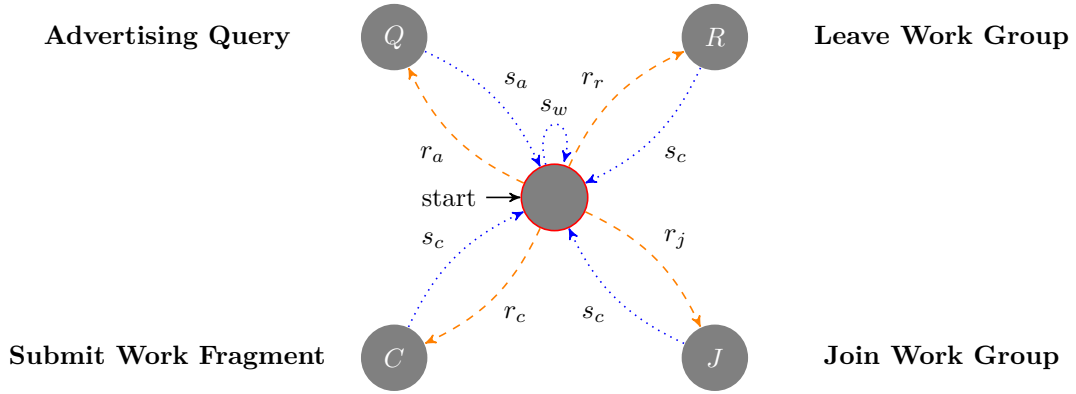
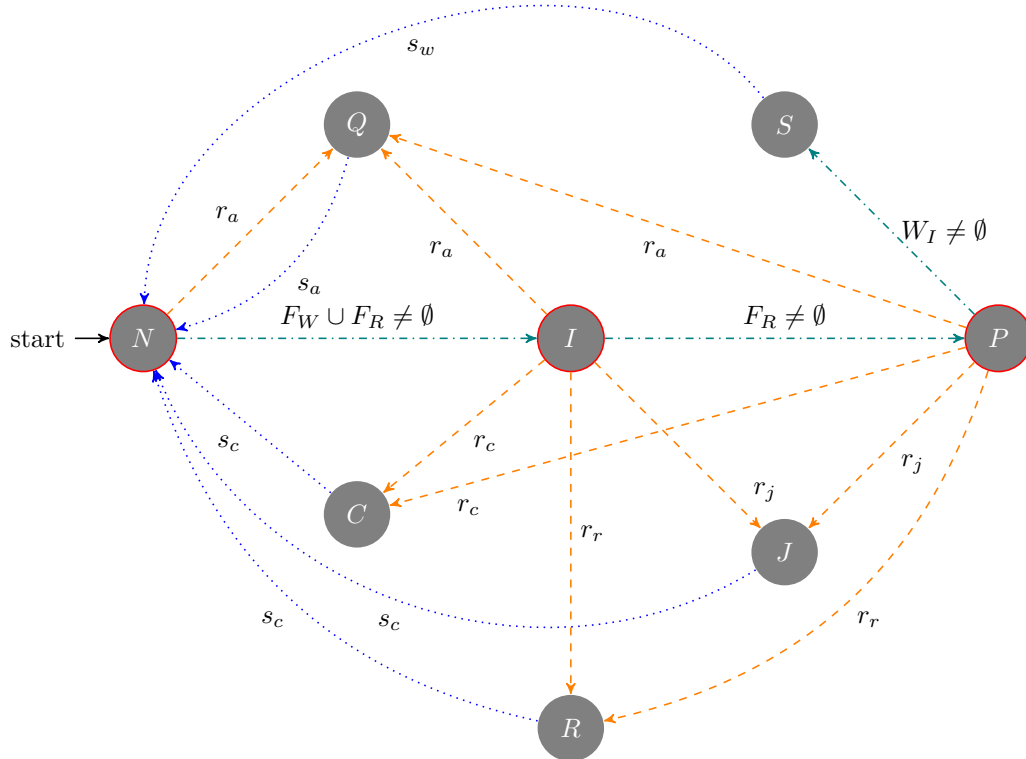


Figure 5.12: A *Worker Node* sends a *Work Group* removal request r_r to a *Work-Issuing Node*. The *Work-Issuing Node* actions the request and sends acknowledgement s_c .

At any time a *Worker Node* can leave a work group by communicating this with the *Work-Issuing Node*. If a *Worker Node* sends a *Work Group* removal request after being sent a *Work Fragment* for evaluation, but before returning the results, the *Work-Issuing Node* can assume the results will never arrive and can freely redistribute the *Work Fragment*.

Work-Issuing Node Interface

These individual behaviours come together to form the complete set of expected behaviour for a DSCS work-issuing node in the *Nosy Nodes* threat model, the public interface into which is given in Figure 5.13. A complete state machine is given in Figure 5.14 including some internal behaviours required to reason about system soundness.

Figure 5.13: Public DSCS *Work-Issuing Node* interface in the *Nosy Nodes* threat model.Figure 5.14: A state machine for prototypical DSCS work-issuing node behaviours in the *Nosy Nodes* threat model. All r transitions are messages from remote worker nodes, and all s transitions responses sent from the work-issuing node.

5.6.4 Properties

The definition of DSCS in the *Nosy Nodes* threat model is relatively sparse as the threat model assumes all nodes act altruistically and none try to subvert the security of the protocol or exploit their peers. Consequently the security arguments for this variation of DSCS are relatively straight forward. As a TCM is an SMUS with further restrictions, the initial arguments presented in this section show that the *Nosy Nodes* variation of DSCS is a valid SMUS in this threat model. Further arguments are then provided to show DSCS also satisfies the further restrictions required of a TCM. The arguments presented in this section are relatively informal, a symptom of the heavy restrictions placed on any adversarial behaviour by the security model.

SMUS Properties

In order to be an SMUS DSCS must satisfy four requirements: *Multi-User Computation Platform* (Definition 5.4.1), *Data Availability* (Definition 5.4.2), *Data Confidentiality* (Definition 5.4.3), and *Data Integrity* (Definition 5.4.4). As the current threat model heavily restricts any adversarial activity proofs of security are considered needlessly verbose, and instead simple arguments are given for DSCS satisfying these required properties.

Being a *Multi-User Computation Platform* requires that users with identities $i \in I$ can read and write to data store V , and launch programs on computational resource. The bijective mapping from nodes in DSCS to identities satisfies the requirement for the existence of $i \in I$. The argument for the existence of a data store is more nuanced, as clearly there is no shared synchronised data store between nodes. Instead, the shared data store is a more abstract object, where nodes reveal parts of their internal data store to each other as they share *Work Fragments*, and then accept data to be written to their internal data store upon receipt of the *Work Fragment* result. Importantly, each node is sovereign over an exclusive part of the data across all nodes, and all data is owned by no more than one node. Access to data is tightly controlled by the owning node, with the only way to read or write to an unowned data store being through the acceptance, completion, and return of a *Work Fragment*. Nodes freely launch programs of their own design, compiled into *Work Fragments*, and due to the security model it can be assumed that all nodes work on the tasks of their peers enthusiastically, and so programs are executed on a shared computational substrate C .

Arguments for *Data Availability*, and *Data Integrity* follow from the tightly controlled access patterns over the data store. The data is always available because the canonical version is stored locally at each node. Similarly, with the only way to write to a peers data store involving the completion and submission of a *Work Fragment* the integrity of which can only be compromised if a node deviates from the security model.

The *Data Confidentiality* property reduces to the strength of an underlying IND-CPA secure LFHE cryptosystem. As all private data is encrypted under an IND-CPA secure LFHE scheme before it leaves the node, and as such violating data confidentiality requires the capability to attack this underlying cryptosystem. Because all private data is suitably tagged, and all data derived from private data inherits this tag, DSCS satisfies *Data Confidentiality*.

Clearly DSCS satisfies the four requirements of an SMUS: *Multi-User Computation Platform*, *Data Availability*, *Data Integrity*, and *Data Confidentiality*. Therefore, in the *Nosy Node* security model DSCS qualifies as a SMUS.

TCM Properties

The class of TCMs imposes further restrictions on SMUSs. These additional properties are: *Distributed SMUS* (Definition 5.5.1), *Decentralised SMUS* (Definition 5.5.2), *Fair Resource Allocation* (Definition 5.5.3), and *Low Longevity Cost* (Definition 5.5.4). The *Distributed* and *Decentralised* properties are self-evident in the definition of DSCS.

Fair Resource Allocation is not enforced by the current definition of DSCS. Instead, a similar capability is an emergent property of the security model. The security model assumed that all nodes engage enthusiastically with the workloads organised by their peers. As such, the resources assigned to each task are dependent on the ratio between the computational burden of a given task and the total computational burden of all in-flight tasks. This, of course, is a slightly different property, but semi-trusted adversaries are unable to exploit this for their own gain without violating the assumptions that they do not maliciously saturate the network with tasks. Because of this deviation DSCS as defined in this section is not a true TCM. Further machinery is required to provide guarantees of *Fair Resource Allocation*, but ultimately this is not a concern in this security model.

The *Low Longevity Cost* property follows from inspection of the DSCS protocol specified in Figure 5.14. At no point does anything need to be saved to long-term storage. No variables persist after the completion of a program. Work groups are temporal and are dissolved as work is finished. The only required long-term storage is the addresses of nearby peers and this can be discarded or pruned, and reconstructed as required. A node can purge its local data store without affecting their ability to rejoin, rediscover, and continue processing *Work Fragments*.

As DSCS has most of the properties given in the paragraphs above: *Distributed SMUS* (Definition 5.5.1), *Decentralised SMUS* (Definition 5.5.2), and *Low Longevity Cost* (Definition 5.5.4). It satisfies all the conditions, except for *Fair Resource Allocation*, required to be classified as a TCM. Given the assumptions of the *Nosy Node* threat model the behaviour of DSCS as defined in this section is close enough to a TCM that it can target the same use-cases as a true TCM. This is because an adversary is unable to exploit the non-TCM behaviours for their own gain.

5.7 DSCS in the Lazy Node Threat Model

In this section the DSCS specification given in the previous section is expanded to maintain the properties of a TCM in the face of a more empowered adversary in the *Lazy Node* threat model. As a reminder, this threat model empowers adversaries to delay or ignore network messages. In DSCS specifically, this manifests as an adversarial node who does not join the *Work Groups* of their peers but still offloads work to a *Work Group* of their own.

Nodes may be motivated to act lazily in order to improve their perceived performance relative to their peers. For example, in an IoT deployment with a variety of competing brands it may not be in the best interests of a device to do work to improve the perceived latency of devices provided by direct competitors. Similarly, by not engaging proactively in the network devices can reduce the power consumed through their operation. This results in an overall worse experience for the user, across all their devices, while improving the perceived performance of these lazy nodes in comparison to devices acting more proactively in the DSCS network.

In order to maintain the TCM properties of DSCS in this new setting a simple trading economy is added to the protocol specification. When a *Work Fragment* is completed the *Work Issuing* node rewards the

Worker node with credit redeemable for an equivalent amount of work. Mechanically, each node maintains an integer for each peer node representing work available in debit or credit at the peer node. The one-to-one correspondence of *Work Fragments* to arithmetic circuits means their execution cost is easily estimated in advance by both parties. This feature, paired with the fact no nodes in this setting act overtly dishonestly, means both nodes will always agree on the balance between them. Between any pair of nodes, a node with a positive balance can always send a *Work Fragment* with a cost less than the accrued credit to the node with the negative balance and expect them to evaluate the *Work Fragment* in a timely manner.

This credit system additionally puts pressure on nodes to have a longer network liveliness, as the liveliness of a node will affect its ability to fulfil their work obligations to their peers. In this setting nodes will not act dishonestly and so will not abandon identities with a negative balance of credit. That being said, there will be some subtle pressure towards favouring the *Work Groups* of nodes with a reliable presence on the network as these can be relied upon to be available to evaluate *Work Fragments* in a timely manner.

The entire credit construction for DSCS is similar in effect to the “tit-for-tat” data exchanges of the BitTorrent protocol. The difference being that in DSCS there is no need to provide an unchoking mechanism, due to the inherent asymmetry between the *Work-Issuing Nodes* and *Worker Nodes*. *Worker Nodes* are encouraged to actively seek out *Work Groups* in order to accumulate credit with their spare computational power for later use. This contrasts to BitTorrent, where this proactive unchoking is impossible as nodes must first receive data to be able to send data. In DSCS devices can effectively unchoke themselves by volunteering for *Work Groups*. In subsequent extensions to DSCS the use of signature schemes and FE constructions is required to enforce honesty in the network participants when handling credits. However, in the *Lazy Nodes* security model adversarial nodes are unable to deviate from the protocol and so there is no need to use additional machinery to enforce expected behaviours.

By adding these techniques participating nodes are encouraged to do work for each other in order to accumulate credits which can be redeemed when they experience heightened load. In effect, for all the work a node is capable of doing credit can be accumulated and then leveraged in a single large low-latency computation which requires a comparable amount of work. This mechanism acts as a store of computational work allowing nodes to defer computational ability they have now until they need it later. Ultimately, a node can leverage as much work from the network as it can in turn provide.

The mechanism for generating and validating work credits is designed such that the burden of storage is independent of the number of credits that have been issued. This design philosophy will become more important as additional cryptographic machinery is added to secure the credit system in the face of an adversary in the *Untrusted Node* security model. For now, the *Work-Issuing Node* issuing credit and the *Worker Node* receiving the credit need only store the latest balance of credit.

Each node has an individual view of the network, including which nodes have a high degree of liveliness. This diversity is intentional and reduces the chance of deadlock by encouraging a variety of jobs prioritised differently and worked simultaneously. This is important for counteracting network stagnation, where each node behaves identically and some *Work Groups* become saturated while others are stagnant. Nodes are free to develop their own mechanisms to make internal decisions on how they interact with the network, the mechanics and implications of such a process are beyond the scope of this thesis, but in this respect diversity will no doubt be advantageous.

To provide a simile encapsulating the mechanisms added to DSCS when moving into the *Lazy Node* threat model, it is like friends writing out and redeeming IoU notes. Each party trusts the other to honour the IoU and not to attempt to produce forgeries, but the existence of the IoUs allow for a fair and equitable balance

Operation	Credit Weight
Integer Arithmetic	1
Ciphertext Addition	50
Ciphertext Fragment Addition	10
Ciphertext Multiplication	100
Ciphertext Fragment Multiplication	20

Table 5.2: DSCS heuristic credit weightings. These heuristics only loosely reflect reality, and the normalised computational burden of each operation is dependent on external factors such as cryptosystem choice and parameters, and availability of hardware coprocessors.

of resource expenditure between the participants.

5.7.1 Extended Protocol Definitions

In this subsection the extended DSCS mechanisms described above are given more formally, once again with interacting state machines. The term *Work-Issuing Node* is used as before, but in many cases it is used interchangeably with *Credit-Issuing Node*. The latter term is used to denote contexts where the node is the source of credit.

Issuing a Credit Reward

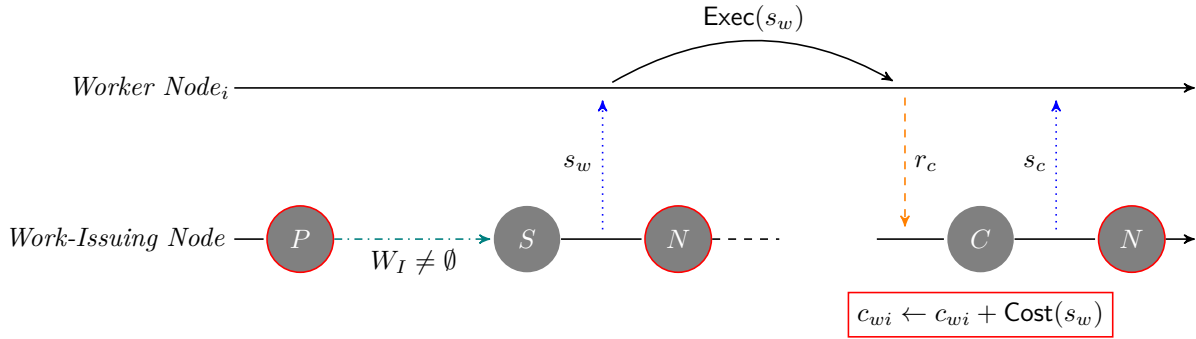


Figure 5.15: An extension to the mechanism for issuing work given in Figure 5.11 to support a credit reward for completed *Work Fragments*. The new addition is highlighted in a red square. The *Work-Issuing Node* keeps an internal counter (initialised to 0) for each collaborating *Worker Node*. When the result of a *Work Fragment* is returned, the *Work-Issuing Node* increases the internal credit counter relevant to that *i*th *Worker Node*, c_{wi} , by the value of the *Work Fragment*, provided by function Cost .

The updated definition for the process of issuing work to a work-group features a final step whereby the *Work-Issuing Node* increases the credit balance of the *Worker Node* as compensation for their effort. As estimating the cost of a *Work Fragment* is easy, a weighted linear combination of the costs of all operations contained within the *Work Fragment*, the *Work-Issuing Node* and the *Worker Node* always agree on the updated credit balance between them. The additional step of updating the credit value is added in the flow shown in Figure 5.15. Example heuristics weights for DSCS *Work Fragment* operations are given in Table 5.2, these should be fixed in advance so each node on the network agrees to their values.

Redeeming Credit

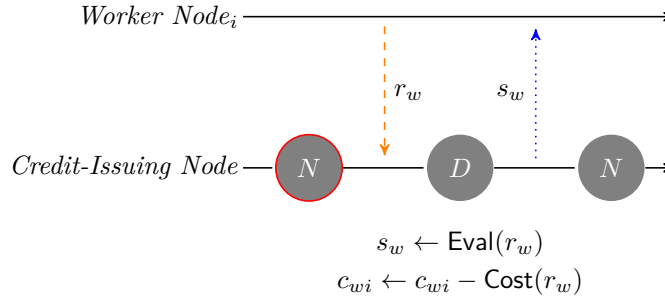


Figure 5.16: A *Worker Node* with a pre-existing positive credit balance sends a *Work Fragment* completion request r_w to the *Credit-Issuing Node* with which it had accumulated credit. The *Credit-Issuing Node* evaluates the *Work Fragment*, decreases the credit total c_w by the appropriate cost, and then returns the results to the requesting *Work Fragment*.

A node which has previous accrued credit as a *Worker Node* can send a *Work Fragment* completion request to the node which issued the credit. The *Work-Issuing Node* evaluates the *Work Fragment*, adjusting the credit balance as necessary, and returns the result to the requesting node. This workflow is shown in Figure 5.16.

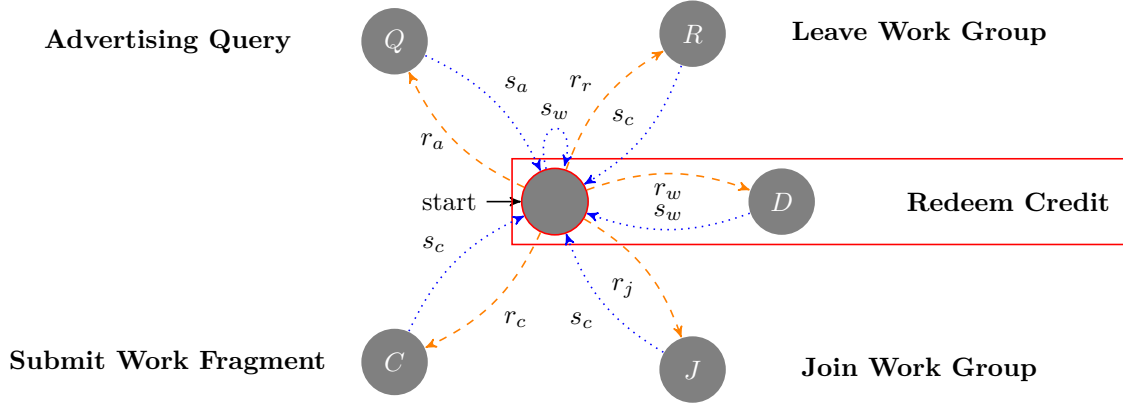
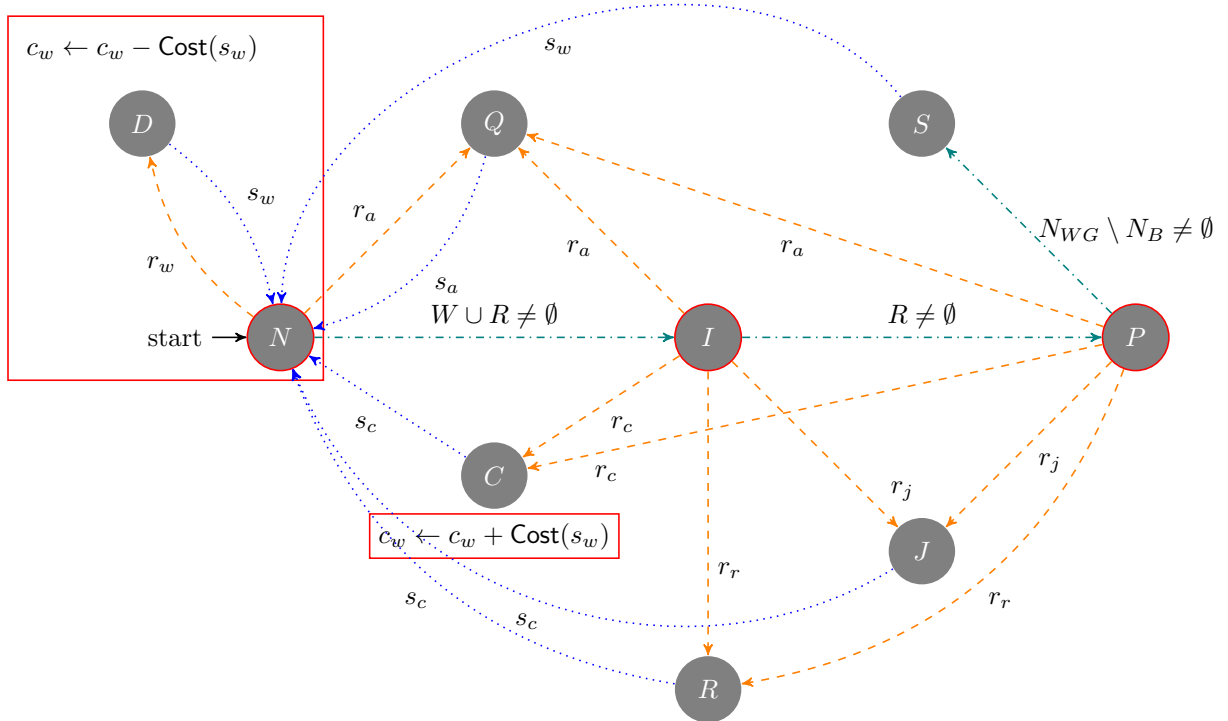
Credit can only be redeemed at the issuing node. This decision was taken as if credit is transferable it becomes possible for a cabal of malicious nodes to mint credit for themselves, for use extorting computational effort out of honest nodes. By limiting credit to use at the issuing node this attack vector is eliminated, but conversely credit becomes somewhat brittle. It becomes possible for nodes to mint credit, and then become unreachable, rendering any minted credit useless. In balance this potential attack can be mitigated by encouraging the credit receiver to spend accrued credit quickly, as any hoarded credit could be rendered useless if the issuing node becomes permanently unreachable. This heuristic pressure has the added benefit of reducing the possibility that a huge volume of credit is suddenly redeemed by multiple nodes at a single credit-issuing node, which would force the credit-issuing node to queue incoming credit-backed requests for work, thereby impacting computational latency.

Work-Issuing Node Interface

The two additional behaviours, **Issuing Credit Rewards** and **Redeeming Credit** augment the interface each node presents to the network, causes changes in the publicly facing interface defined originally in Figure 5.13 and the heuristic inner mechanisms shown in Figure 5.14, which are redefined in both Figure 5.17 and Figure 5.18 for the *Lazy Node* security model. As in the previous threat model (Figure 5.14), the more verbose state transition diagram given in Figure 5.18 serves as a heuristic implementation guide and aids in subsequent security proofs.

5.7.2 Properties

As in Section 5.6, arguments will be put forward that this version of the DSCS protocol satisfies the properties of a TCM, this time in the *Lazy Node* threat model. The difference in adversarial capability when moving into the *Lazy Node* model is that the adversary will not proactively engage with the network if they could conceivably perform better alone. This could empower the adversary to exploit more altruistic nodes for

Figure 5.17: Public DSCS *Work-Issuing Node* interface in the *Lazy Nodes* threat model.Figure 5.18: A state machine for DSCS *Work-Issuing* node behaviours in the *Lazy Nodes* threat model with additional heuristic internal state transitions.

their computational power. As in Section 5.6, each SMUS and TCM property is approached in turn and justifications given for satisfying the definition in this threat model.

SMUS Properties

Arguments pertaining to the *Multi-User Computation Platform* property, as well as the *Data Availability*, *Data Integrity*, and *Data Confidentiality* properties are unchanged from the previous section. These foundational arguments about system capabilities are obviously satisfied by this protocol with only a small amount of additional machinery supporting a system for the exchange of credit. Given the minor differences between the DSCS as defined for the *Lazy Node* and *Nosy Node* threat models, it is clear that DSCS still satisfies the properties of an SMUS.

TCM Properties

As in the previous security model, the *Distributed SMUS* and *Decentralised SMUS* properties of a TCM are self-evident in the definition provided for DSCS. However, an argument needs to be made for *Fair Resource Allocation* in the face of the newly empowered adversary, and how arguments about *Low Longevity Cost* are affected by the newly added credit machinery.

In order to reason about the computational power available to nodes, consider the abstracted true metric of the computational throughput of network participants $c \in C$. Where c is the momentary computational power averaged across a hypothetical network running until infinity. With this definition of computational power a less powerful device running for longer can be equivalent to a more powerful device, or many devices, running for less time. Within the abstract network model given by this thought experiment, assume that each node produces *READY Work Fragments* based on some unknown, and unimportant, random distribution.

Lemma 5.7.1 (Fair Resource Allocation). In the *Lazy Node* security model, for DSCS node (i_1, c_1) the networked computational resources allocated to their computational tasks, c'_1 , is at least proportional to their own local computational resource c_1

$$c'_1 \propto c_1 \quad (5.5)$$

Proof. In this security model nodes are empowered to act lazily if it benefits them. Should working independently provide more utility than collaborating with peers, the network will suffer and all participants will be worse off because of it. To encourage proactive engagement with the network, a credit system was introduced as part of the extensions to DSCS in this security model.

The argument central to establishing the *Fair Resource Allocation* property in this setting is that by engaging with the protocol nodes can do at least as well as if they acted completely independently, resulting in the situation where $c'_1 = c_1$. By proving a node can do no worse than this by interacting with the network it must be the case that c'_1 is at least proportional to c_1 .

Nodes are rewarded with credit for evaluating *Work Fragments*. The value of the credits is set based on a linear combination of the heuristic weights of the operations within the evaluated *Work Fragment*. Accumulated credit can be redeemed for a quantity of computational work with equivalent heuristic costs to the original work done to amass the credit. Nodes are able to spend computational effort now to generate credit to leverage network computational effort later. The security model ensures no node discards an identity with a negative credit balance. It is clear that by engaging with the network the node can do no worse than

if they operated in isolation. As such, at the very least

$$\therefore c'_1 \propto c_1 \quad (5.6)$$

□

There is a possibility that a sufficiently motivated malicious node could exploit incongruencies between the heuristic credit weights for instructions and their actual cost to get peers to do overcosted operations in exchange for undercosted operations thereby leveraging more computational power than they offer. This capability is contingent on being able to craft the *Work Fragments* that are sent to them. Yes, they have the capability to discard fragments that have been delegated to them but only to find other *Work-Issuing Nodes* with which to find work. The malicious node will have no control over the *Work Fragments* they are sent from this new node. The probability of mounting an attack of this nature is negligible if the distance from truth of the heuristic weights is small enough that the computational burden of evaluating, discarding, and reprioritising *Work Fragments* is relatively large in comparison to any possible gains. Unfortunately, an incongruence between reality and the heuristic cost of a work fragment is unavoidable with a network of non-homogeneous nodes and a single shared cost function.

Although, given differing computational abilities and varying access to hardware accelerators, different nodes will be better suited to different operations. This fact should be encouraged and actively exploited in further research. Economic pressures will see specialised nodes seek out well-suited work to earn credits and pay for workloads to which they are ill-suited. The mechanics of such a process and the implications on this work are beyond the scope of this thesis.

The addition of the credit system allows DSCS to fully qualify as a TCM for the first time, as now it satisfies the requirements of the *Fair Resource Allocation* property.

Unfortunately, the addition of a credit system to the DSCS protocol has the potential to cause the long-term storage requirements on nodes to exceed that which would be tolerated by the *Low Longevity Cost* property. However, due to the careful design of the system, the data requirements scale with local network size and not network age. The following proof is not difficult, but it serves as important context as additional cryptographic machinery is added to the credit system in the forthcoming section to maintain security against more empowered adversaries in the *Untrusted Node* security model.

Lemma 5.7.2 (Low Longevity Cost). Assuming that the number of nodes on the network at any given time is independent of the runtime of the network, in the *Lazy Node* security model, the local storage requirements to actively participate in a DSCS network is independent of the runtime of the network.

Proof. The data footprint on a single DSCS node required to support a single node-to-node credit total consists of a single integer representing the total credit balance between the nodes. This data requirement is fixed and does not grow over time as the network operates.

Each node must maintain a credit data store consisting an integer for each other node they interact with. At worst, this is linear in the size of the network.

$$\mathcal{O}(|N|) \quad (5.7)$$

Therefore, assuming that the number of nodes in the network is independent of the lifetime of the network, the per-node data storage requirements are independent of the runtime of the network. □

Theorem 5.7.3 (TCM in Lazy Node Model). In the *Lazy Node* security model the extended version of DSCS presented in this section satisfies the properties of a TCM.

Proof. DSCS, as presented in this section, has been proven to satisfy the requirements of an SMUS, and the further restrictions required by the *Distributed SMUS*, *Decentralised SMUS*, *Fair Resource Allocation*, and *Low Longevity Cost* properties. It therefore satisfies the conditions of a TCM in the *Lazy Node* security model. $\square$

5.8 DSCS in the Untrusted Node Threat Model

The *Untrusted Node* threat model empowers adversarial nodes to act arbitrarily maliciously for personal gain. Taking DSCS as defined in the previous section, there are four major exploitation vectors possible for malicious activity:

1. **Falsification of results**, where a *Worker Node* returns incorrect results for the evaluation of a *Work Fragment*.
2. **Falsification of credits**, where a *Worker Node* attempts to redeem unearned credit, or a *Credit-Issuing Node* denies an attempt to redeem honestly earned credit.
3. **Unfair exchange of credits**, where a *Worker Node* will try to gain credit without releasing the results of a *Work Fragment*, or a *Worker-Issuing Node* will attempt 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.

When attempting a **falsification of results** attack a threat actor may try to do one of two things: do less computational work for credit, or attempt to subvert the work-issuing node's computation by constructing bespoke incorrect results. These are labelled here as *Lazy Falsification* and *Strong Falsification* respectively. *Lazy Falsification* is easier to defend against, it occurs when an adversary wishes to get a credit reward without performing the requisite work. In this case an adversary attempts to generate convincing looking results by exerting no more computational effort than it would cost to generate the result honestly. This adversary is clearly strongly computationally bounded by the effort required to evaluate a given *Work Fragment*, and is therefore relatively weak. *Strong Falsification* occurs when an adversarial entity wishes to influence the output of a function being calculated by a peer by specifically crafting a bespoke fake result for a delegated *Work Fragment*. They may be motivated to force a certain result in the calculations of the *Work-Issuing Node* for their own gain. The adversary is willing to invest disproportionate computational effort to construct the falsified results and as such they are no longer bounded by the computational effort required to evaluate a delegated *Work Fragment*. As such, the adversary is polynomially bounded in some security parameter λ .

In the second category of attack, **falsification of credits**, an adversary is interested in using forged credits to leverage the computational power of their peers. By making the conservative assumption that generating a forged credit of a high value is as difficult as generating one of low value, the adversary must be willing to invest a substantial amount of computational effort to their forgery. As such, the adversary is polynomially bounded in some security parameter λ . In DSCS as defined for the *Lazy Node* threat model, in Section 5.7, there is no recourse for an honest node who is wrongfully denied in an attempt to redeem

honestly earned credit. Nor is there any way for a third party to know who is acting honestly when two nodes disagree on their shared credit balance. Just as an adversarial node may try to redeem credits that have never been issued, an adversarial node may reject an attempt to redeem credit that was honestly earned by a peer. In previous definitions of DSCS it is unclear to an external observer which node is acting honestly. To harden the scheme in this regard a token is exchanged which is able to provide a source of truth to any third party peers. Without this extension, the only way to prove ones own honesty is to store all network messages and provide them to a third party auditor, this would violate the *Low Longevity Cost* property.

When considering mutually untrusting peers exchanging credit, the fair exchange of credit for results is a natural problem. The third category of attack concerns itself with the **unfair exchange of credits**. There are two points of tension in the DSCS protocol when fair exchange becomes relevant; first is when a node is rewarded with credit for completed work, the second is when a credit balance is decreased after completing work. In both cases one node does not wish to release the results of an evaluated *Work Fragment* until the credit total is adjusted, and the other node does not wish to adjust the credit total until the results have been released.

The fourth and final category of adversarial activity involves nodes who violate the protocol in some way and then attempt **clean identity generation**. This can occur after detected attempts to mount attacks captured in the other categories and gets blacklisted from interacting with their peers. The blacklisted node then decides to shed their contaminated identity and generates a clean one. In this way they can begin exploiting their peers anew.

The modifications to DSCS presented in the following subsections are cumulative, and each modification aims to add strong defensive measures against one of the categories of adversarial activity. In some cases the defences against one category of attack are mutually exclusive to another and resist composition. Such cases lead to interesting research questions to guide further work, and are explored briefly as they appear. Explicitly, VC and FE resist naive composition, so at the moment an implementer must chose between defending against **falsification of results** or the **unfair exchange of credits**.

5.8.1 Defences Against Falsification of Results

The first category of attacks involved falsification of results, both *Lazy Falsification* and *Strong Falsification*. In *Lazy Falsification* an adversary is looking to earn credits by doing less work, and in *Strong Falsification* an adversary is looking to invest computational effort to create forgeries to undermine the computations of their peers. The difference between the two is the motivations of the adversary imply different computational bounds, with the first implying a significantly more computationally restricted adversary. There are natural applications for VC protocols in hardening DSCS against both of these classification of attacks.

In *Lazy Falsification*, where the adversary is trying to minimise the amount of effort expended to earn credit, there are a number of appropriate solutions in domains relating to VC:

1. Economic approaches to VC [19, 65]
2. Embedding known test kernels in outsourced computations

In economic approaches to VC fines are levied against misbehaving nodes and by checking a minimal set of incoming results an adversarial actor will, on average, do worse when cheating as compared with acting honestly.

An alternative approach to defend against the more restricted adversary in *Lazy Falsification* is to create and embed test kernels within existing applications. These can be randomly generated at compile time with

known answers such that when results of outsourced computation are received they can be checked and rejected if incorrect. Circumventing this requires first executing the *Work Fragment*, which would exceed the computational bounds on this weaker adversary.

Defending against the more powerful adversary, in the *Strong Falsification* problem, requires using number theoretic VC constructions. There are a wide variety of constructions available, some contemporary selections include: Pinocchio [95], Bulletproofs [34], Libra [111], or the VC scheme presented in Chapter 3. The VC scheme presented in Chapter 3 is particularly well suited to DSCS due to the efficiency in the one-shot execution setting, the Prover-optimality, and the performance when working with arithmetic over large integers - as would be the case when handling most LFHE ciphertexts. Also, this construction requires no extra machinery on the *Worker Node* as all modifications are done on the *Work-Issuing Node* and only within the steps taken to process incoming results. All known alternative number theoretic VC constructions perform poorly on one-shot programs and instead obtain great results when verifying programs that are executed repeatedly with different inputs. In order to make the most general DSCS definition, the VC presented in Chapter 3 appears to be the best candidate.

Regardless of the choice of VC construction, DSCS uses VC as a black box. When using number theoretic VC schemes there is no need to deploy the economic style of VC schemes nor test kernels. In order to defend against strong adversarial nodes number theoretic VC is used, as it provides the strongest verification guarantees.

Verifying the Results from Outsourced Work Fragments

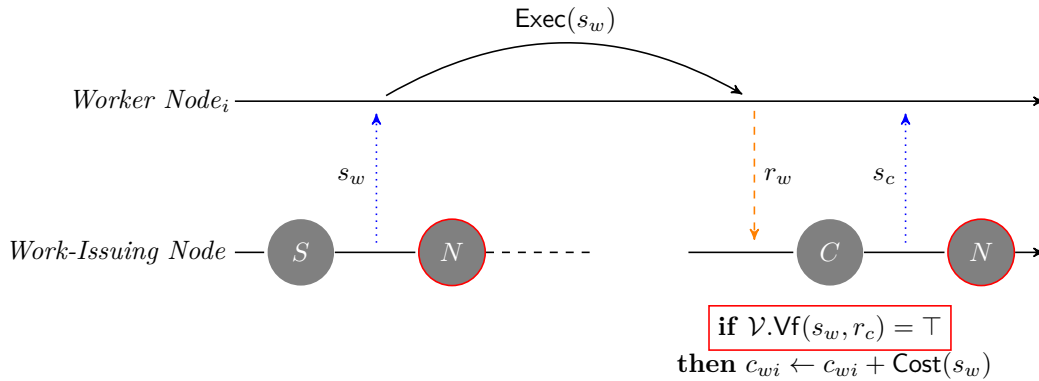


Figure 5.19: An extension to the mechanism for issuing work given in Figure 5.15 to support the verification of returned results, the new behaviour is highlighted in a red box. The black box VC scheme used here is assumed to have a minimal implementation, such as the VC scheme presented in Chapter 3, but this can be easily made more general with the addition of the standard VC functions for problem generation and evaluation.

As shown in Figure 5.19, a VC scheme is inserted as part of the workflow of outsourcing *Work Fragments* and receiving results. Preprocessing required for some VC schemes, *KGen* and *ProbGen*, can be embedded within the steps taken in state *S*. Following successful result verification credits will be issued as before.

5.8.2 Defences Against Credit Falsification

The credit falsification problem is mostly one of auditability. Credit disputes occur between two peers which are convinced of a different shared credit balance, and a third party cannot validate the correctness of either

$\text{CreditGen}_{\text{sk}}^S(i, d, v_n, v_b)$
1 : $c_v \leftarrow (v_b + v_n, i, d + 1)$
2 : $c_s \leftarrow \mathcal{S}.\text{Sign}_{\text{sk}}(c_v)$
3 : return (c_v, c_s)

Figure 5.20: DSCS credit generation, given signing secret key sk , *Worker Node* public key i , previous counter value d , new credit value v_n , and previous credit balance v_b .

$\text{CreditVal}_{\text{pk}}^S(r_v, r_s, d)$
1 : $(v, i, d') \leftarrow r_v$
2 : if $d' \geq d \wedge \mathcal{S}.\text{Vf}_{\text{pk}}(r_v, r_s) = \top$ then
3 : return $\top$
4 : endif
5 : return $\perp$

Figure 5.21: DSCS credit validation. Given the credit token r_c and corresponding signature r_s returns true if the signature is valid and the embedded counter is greater than equal to the counter on disk.

claim. Currently nodes must maintain a complete log of all protocol messages to provide evidence to a third party of their honesty. This is possible because all messages are signed by the sender, and so it becomes easy to validate the honesty of the honest party. Unfortunately, this requires parties keep exhaustive records of network messages. Regardless, dishonest parties can still falsify information via maliciously omitting messages. One mechanism to add auditability to a decentralised system like this is to produce a cryptographic source of truth which can be used to convince any third party node of a node's honesty. The existing credit construction can be extended such that both parties must agree on the true credit total, or be proven a dishonest actor. Before the extensions are applied a third party wouldn't know who to trust if one node was accusing another of cheating. With the additional machinery it is possible for the wider network to act as an adjudicator, the specifics of which are beyond the scope of this thesis.

The extension to the credit machinery introduces a token which can be given to any third party in order to prove another node is acting dishonestly and cause appropriate network-wide denunciation. Each time the credit balance is updated the node which is not benefiting from the balance change, either because they are reducing their credit or because they are increasing the credit of a peer, cryptographically signs the updated balance and passes this to the credit beneficiary. This signed token, c_s , also includes a strictly increasing counter. Should the honesty of a node be brought into question they must merely show they are adhering to the signed token with the highest counter value.

In order to generate a credit token the *Credit-Issuing Node* node bundles the newly-awarded credit balance $v_n + v_b$ with the public key of the receiving node, i , and an counter value which is strictly larger than the previous counter value, d . This structure, c_v is then signed by the *Credit-Issuing Node* to produce signature c_s and both c_v and c_s are sent to the node receiving credit. This credit signing construction is defined in Figure 5.20. This construction has the added benefit that the node that issues the credit does not necessarily need to hold onto it any more. Their situation is strictly worse than before they issued the credit and so they can leave the burden of proof to the node they issued credit to. Instead, they should keep any new credit token on disk, should a remote peer attempt to redeem an outdated credit token.

Credit validation is similarly straight forward. Any network node can validate a credit as genuine by

Redeeming Cryptographic Credit Token

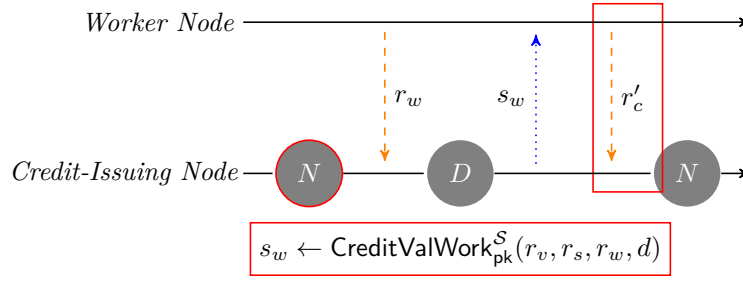


Figure 5.24: Extension to the mechanism for credit redemption in Figure 5.16 to support a cryptographic token of the credit balance.

The extended credit redemption workflow is demonstrated in Figure 5.24. It includes extensions to support a cryptographic credit token. The redeeming node must generate a credit token reflecting the updated balance when they receive the results of their evaluated *Work Fragment* via *CreditGen*, in Figure 5.20.

5.8.3 Defences Against The Unfair Exchange of Credit

Emphasising the importance of the security of the cryptographic token leads into an easily mountable attack exploiting unfair exchange. A straight forward attack can target whoever goes first in the exchange of results for credit. In which the node going second intentionally neglects to uphold their half of the exchange and the first node goes without their credit reward. This can occur both when credit is redeemed or awarded and cannot be solved by switching which party goes first in the protocol.

If the credit is sent before the results of a work fragment evaluation those results may never arrive. If the results are proffered first, the credit may never be awarded. FE schemes are intended to solve this problem in order to support trading digital goods, where the order of exchange opens a vulnerability for whoever goes first. The applicability of these schemes in view of this vulnerability are immediately obvious - the exchange of credit for results must be mediated by an FE scheme. Unfortunately all previous FE constructions with the *Strong Fairness* property required a third party arbitrator, a condition which would violate the *Decentralised SMUS* property of a TCM.

Fortuitously, Chapter 4 introduced a first-of-its-kind two party FE construction with the *Strong Fairness* property. With two party FE any two parties can exchange digital goods if and only if each party does not cheat, without access to a third party mediator. Any transaction in the DSCS protocol in which the order of events offers the possibility of foul play can be secured by performing the exchange over a two party FE protocol.

There is, unfortunately, a catch - composing VC schemes and FE schemes is not trivial. During VC the payment must be awarded after verification of results, whereas during FE the payment and verification happen simultaneously. The naive use of FE would allow payment for unverified work. Without further research, it appears DSCS deployments must choose between VC and FE. There is no reason to believe VC and FE are irreconcilable, as ZKP constructions can provide the required guarantees. The problem lies in the performance impact on VC and the *Outsourceable* requirement. Further work is required to explore the composition of these two protocols.

It is worth mentioning that the definition of FE includes the abstract function *Desc* which is used to provide a description of the data being exchanged. This description is binding, and if data being exchanged

does not satisfy the Desc agreed upon in advance by both parties the protocol fails and neither party receives anything. By trivially setting Desc to evaluate to “The correct results of work fragment w ” the VC scheme and FE scheme can be combined. The process by which one would construct such a protocol while maintaining the strict bounds on computation required by a VC scheme is not clear and requires further exploration.

Fair Exchange of Credit for Results of Outsourced Work Fragments

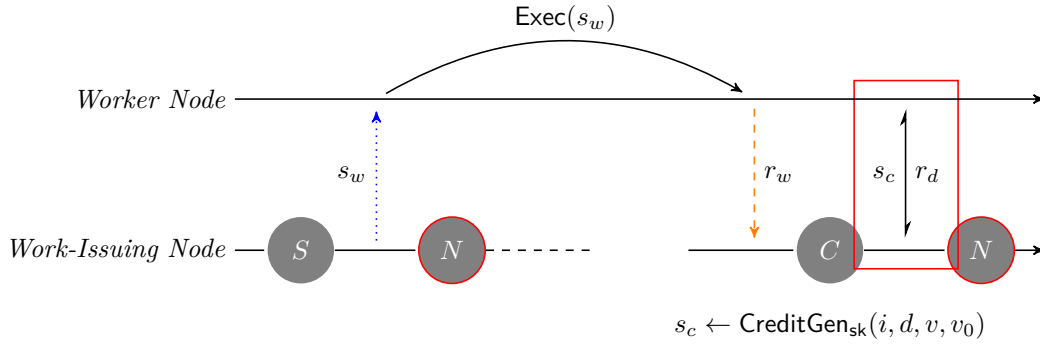


Figure 5.25: Using an FE scheme to mediate the exchange of credit tokens and *Work Fragment* results.

The extensions to the outsourcing *Work Fragment* workflow with cryptographic credit tokens and an FE is shown in Figure 5.25.

Fair Exchange when Redeeming Credit

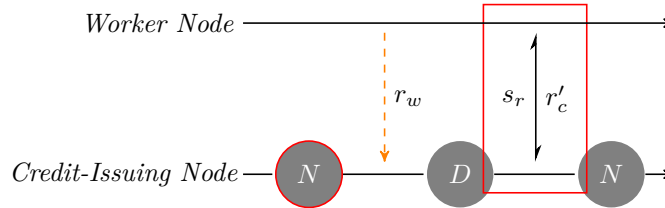


Figure 5.26: Mediating the exchange of results for an updated credit balance when redeeming credit.

In Figure 5.26 two party FE is used to mediate the exchange of a cryptographic credit token for the results of evaluating a *Work Fragment* when redeeming credit.

5.8.4 Defences Against Identity Regeneration

In order to deter the generation of new clean identities, identity generation must carry some heavy cost. The current version of DSCS already exerts a subtle pressure against targeted identity regeneration through the inclusion of a credit system. In the current system nodes need credit to have a degree of agency, and so a new node may need to spend time donating their compute power before they are able to leverage their peers to their full potential. This process takes time and computational effort. Unfortunately, however, this defence is a relatively weak guarantee of node behaviour, and ideally identity regeneration attacks should reduce to some computationally intractable problem. Unfortunately, producing a strong defence in this setting remains

an open problem. The remainder of this subsection details the mechanisms of the heuristic defences already existing in DSCS.

DSCS as it already exists encourages nodes to prioritise collaborating with long-lived network nodes, as they have a proven track record of providing and rewarding computational credit. Newer nodes will not be trusted to award credit, but they can easily “unchoke” themselves by first earning credit which they can later spend on their own processes. In this way long-lived network identities are rewarded by becoming trusted credit generators, and newer identities are not put into a position where they reward credit they are unable to later accept.

One problem is that a sufficiently patient adversary could maintain an honest network presence for long enough to be a trusted generator of credit. At which point they wait until the credit they owe to the network reaches some high value and then they disconnect, never to be seen again. In this way a node is able to put in a non-negligible amount of computational effort to undermine the credit system.

Nodes that know this is a possibility will avoid hoarding credit, as such behaviour will leave themselves vulnerable to the trusted generator of credit disappearing. As such, nodes should spend credit quickly. This is intended behaviour and contributes to a healthy distributed computation network, as it also has the effect that no individual node will be faced with a huge computational task backed by a hoard of accumulated credit. This style of fluid redeeming and generating credit aligns well with the intended setting of IoT devices. These often have small intermittent tasks which run on a regular schedule. Unfortunately, the defences to identity regeneration attacks are heuristic at best.

Thankfully, as credit is only redeemable at the point of generation there is no possibility of two colluding nodes gifting each other credit which can be redeemed at an honest node. Any future attempts to extend the credit scheme to support different generation and redemption nodes must take this additional attack vector into account.

A solution could be constructed where generating a network identity was made arbitrary computationally difficult. Nodes would have to invest a configurable amount of computational effort to generate a random identity which fits some predefined constraints. At best this would produce a rate-limiting effect on malicious identity generation. Similarly, the protocol could disregard the decentralised property entirely and delegate identity generation to a central authority trusted to generate one and only identity per node. In an IoT setting this could occur as part of a workflow wherein a home owner enrolls a new device.

5.8.5 DSCS State Machine in an Untrusted Network

As previously discussed a choice has to be made whether to extend DSCS with VC or FE as they are not naively composable. Where the new FE scheme, as presented in Chapter 4, requires a large number of rounds to successfully terminate, VC schemes have, by their *Outsourceable* property, incompatible constraints on performance. The threat presented by unverified computation is also more severe than that of unfair exchange, because a party willing to cheat on the exchange of credits is likely to cheat on the correct evaluation of a work fragment. As such, until constructions are created that allow the composition of VC and FE, VC has been selected as the most important security extension. The public interface is unchanged from Figure 5.17, a state machine with internal heuristic is given by Figure 5.27. The changes over previous iterations include:

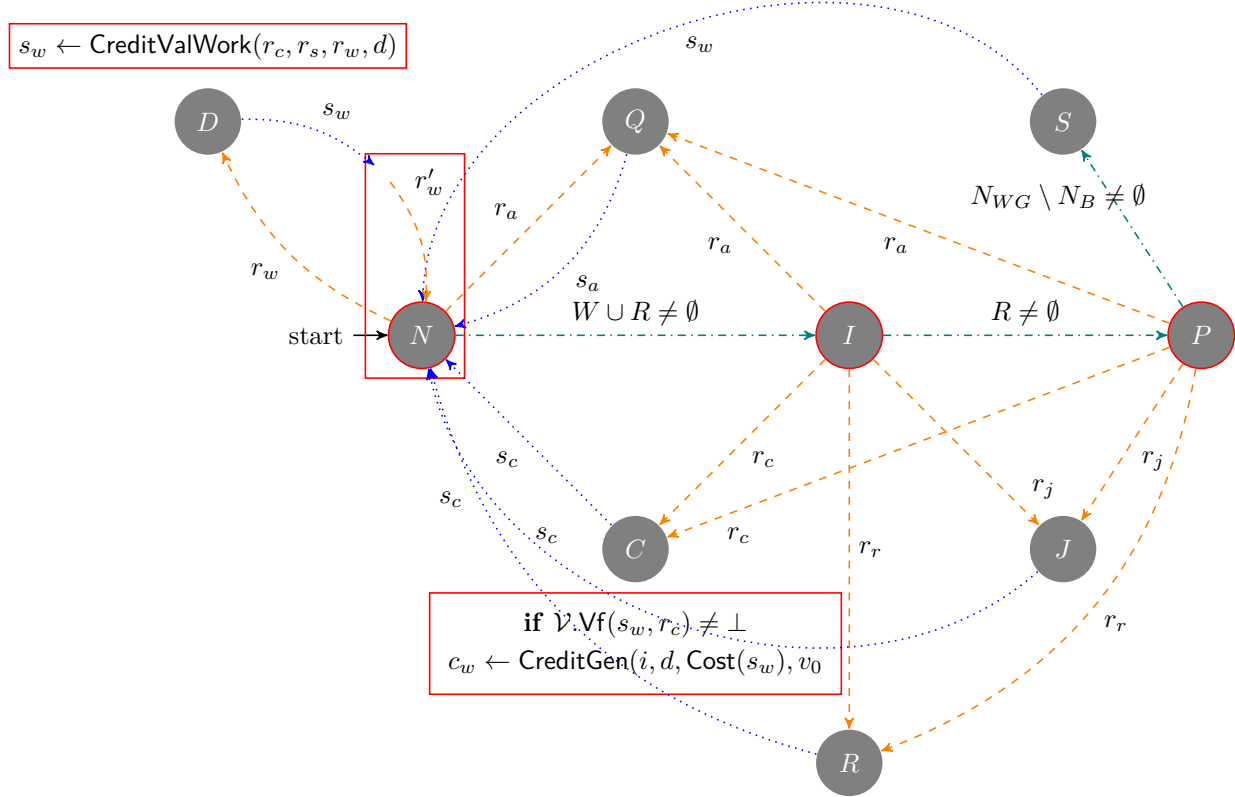


Figure 5.27: A state machine for DSCS *Work-Issuing* node behaviours in the *Untrusted Nodes* threat model, deploying cryptographic tokenised credits and VC, with additional heuristic internal state transitions.

- The inclusion of VC to defend against the **falsification of results**.
- The extension of the credit system with a cryptographically backed token system to defend against the **falsification of credits**.

FE defence against the unfair exchange of credit are omitted due to the incompatibility with VC. The heuristic defences against **identity regeneration** are unchanged from previous versions.

5.8.6 Properties

The following proofs and heuristic arguments will show that DSCS, as defined in Figure 5.27, satisfies the TCM definition in the *Untrusted Node* security model. This leads to obvious attacks against the exchange of credit for work, but the integration of FE and VC is beyond the scope of this thesis, and left for further work.

SMUS Properties

As before, arguments for **Multi-User Computation Platform**, **Data Availability**, and **Data Confidentiality** are carried over from the *Lazy Node* threat model. Each of these properties is either self evident,

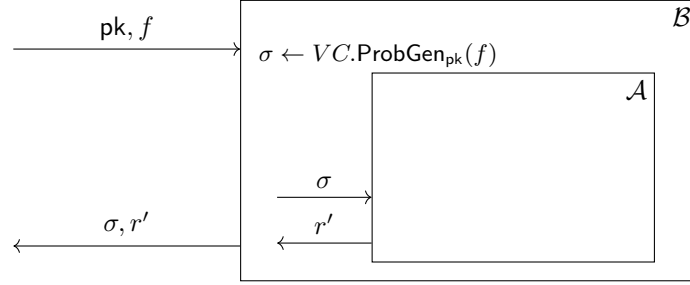


Figure 5.28: Adversary $\mathcal{B}$; capable of winning the `VerifyGame` security game as defined in Figure 3.2, violating the *Soundness* property of an underlying VC scheme given adversary $\mathcal{A}$, capable of producing forged results to computations that evade DSCS detection.

or is still computationally intractable for an adversary to circumvent in the previous threat model. However, the **Data Integrity** property requires more examination, as suddenly adversarial nodes are able to deviate from expected behaviours in order to corrupt data, by submitting incorrectly evaluated *Work Fragments*. To prove DSCS satisfies the **Data Integrity** property in the face of the newly empowered adversaries an intermediate lemma is provided relating to the ability of an adversary to submit incorrect results to *Work Fragments*, which is then used to prove the scheme has the *Data Integrity* property.

Lemma 5.8.1. Provided that DSCS is constructed with a VC scheme with the *Soundness* property, the probability of a node successfully submitting a forged result r' as the output to the evaluation of *Work Fragments* w , such that $r = \text{Eval}(w)$ is the correct result, and $r \neq r'$ is negligible:

$$\Pr[r' \neq r \wedge VC(r') = \top] \leq \text{negl}(\lambda) \quad (5.8)$$

Proof. As all incoming results from outsourced *Work Fragments* are verified via a VC scheme with the *Soundness* property, attempting to submit a result that is not correct is equivalent to directly attacking the underlying VC scheme. A trivial reduction can be made from an adversary capable of tricking the *Work Issuing* node into accepting an incorrect result and an adversary capable of forging results such that it breaks the *Soundness* property of a VC scheme. An illustration of this trivial reduction is given in Figure 5.28.

$$\Pr[r' \neq r \wedge VC(r') = \top] \leq \text{Adv}_{\mathcal{A}}^{\text{verify}}(\lambda) \quad (5.9)$$

From the definition of VC *Soundness*, in Definition 3.2.2:

$$\text{Adv}_{\mathcal{A}}^{\text{verify}}(\lambda) \leq \text{negl}(\lambda) \quad (5.10)$$

$$\therefore \Pr[r' \neq r \wedge VC(r') = \top] \leq \text{negl}(\lambda) \quad (5.11)$$

□

Lemma 5.8.2 (Data Integrity). In the *Untrusted Node* security model, a node with identity i has a negligible probability of being able to change data d with key k in key-value data store V if $(i, k) \notin A$:

$$\Pr[V[k] \leftarrow d'] \leq \text{negl}(\lambda) \quad (5.12)$$

The ability to write data is taken to mean the ability write data arbitrarily, deviating from any pre-

approved data manipulations as stipulated by a data-owner.

Proof. Within DSCS, when $(i, k) \notin A$, the data keyed by k is not owned by the node with identity i . The only opportunity for nodes to influence data owned by other nodes is when evaluating a *Work Fragment* that has been delegated to them. By Lemma 5.8.1 the probability of being able to affect data beyond the scope of the what has been approved is negligible:

$$\Pr[r' \neq r \wedge VC(r') = \top] \leq \text{negl}(\lambda)$$

Therefore, the probability of successfully modifying data when $(i, d) \notin A$ is:

$$\Pr[V[k] \leftarrow d'] \leq \text{negl}(\lambda) \quad (5.13)$$

□

Theorem 5.8.3 (SMUS in Untrusted Node Model). In the *Untrusted Node* security model DSCS is an SMUS.

The arguments for a *Multi-User Computation Platform*, *Data Availability* and *Data Confidentiality* are carried over from the previous security model. Lemma 5.8.2 proves DSCS has the *Data Integrity* property. Therefore, DSCS is an SMUS in the *Untrusted Node* security model.

TCM Properties

The *Distributed SMUS* and *Decentralised SMUS* properties were established as self-evident in both the *Lazy* and *Nosy Node* security models. This fact is unchanged with the extensions required by the *Untrusted Node* security model. However, new arguments need to be made for *Fair Resource Allocation* and the *Low Longevity Cost* of DSCS in the face of a more empowered adversary.

In the remainder of this subsection proofs will be provided to show that DSCS satisfies these additional TCM properties. In order to reason about *Fair Resource Allocation* an intermediate lemma (Lemma 5.8.4) shows forging DSCS credits is computationally intractable. A strong notion of security against credit falsification can be generated by considering a security game shown in Figure 5.29. In this security game adversarial actor must produce a valid credit without having access to the signing key used, but with access to an oracle, $\text{OCreditGen}_{\text{sk}}$, capable of generating valid credits to their specification signed under the correct key. This game carries the stipulation that the forged credit submitted by the adversary cannot be generated by the oracle. The fact that credit falsification is computationally intractable is then used in the *Fair Resource Allocation* proofs.

Lemma 5.8.4. The advantage of a probabilistic polynomially bounded adversary, $\mathcal{A}$, in the credit forging game, as defined in Figure 5.29, is at most that of the advantage of a probabilistic polynomially bounded adversary in the EUF-CMA security game for the signature scheme used to construct DSCS:

$$\text{Adv}_{\mathcal{A}}^{\text{credit}}(\lambda) \leq \text{Adv}_{\mathcal{B}}^{\text{euf-cma}}(\lambda) \quad (5.14)$$

Proof. Given adversary $\mathcal{A}$ with non-negligible probability of winning *CreditGenGame* as defined in Figure 5.30, it is trivial to construct adversary $\mathcal{B}$ with non-negligible probability of winning the EUF-CMA security game as given in Figure 5.31.

$$\therefore \text{Adv}_{\mathcal{A}}^{\text{credit}}(\lambda) \leq \text{Adv}_{\mathcal{B}}^{\text{euf-cma}}(\lambda) \quad (5.15)$$

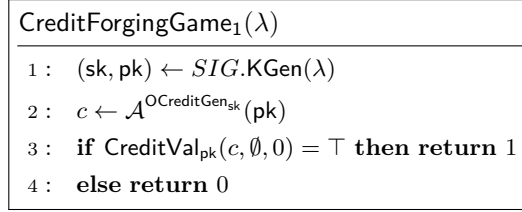


Figure 5.29: The credit forging security game. A pair of keys are generated for signature scheme SIG , adversary $\mathcal{A}$ is given access to the public key and an oracle able to provide valid credits to their specification. The adversary wins if they are able to provide a valid credit that was not signed by the credit generation oracle.

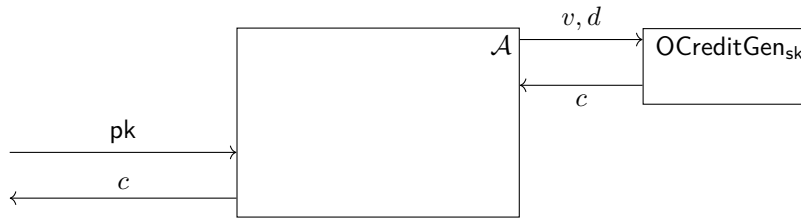


Figure 5.30: Adversary $\mathcal{A}$, capable of winning the CreditForgingGame as detailed in Figure 5.29. Given the availability of credit generation oracle, when provided a public key pk the adversary is able to generate forged credit c .

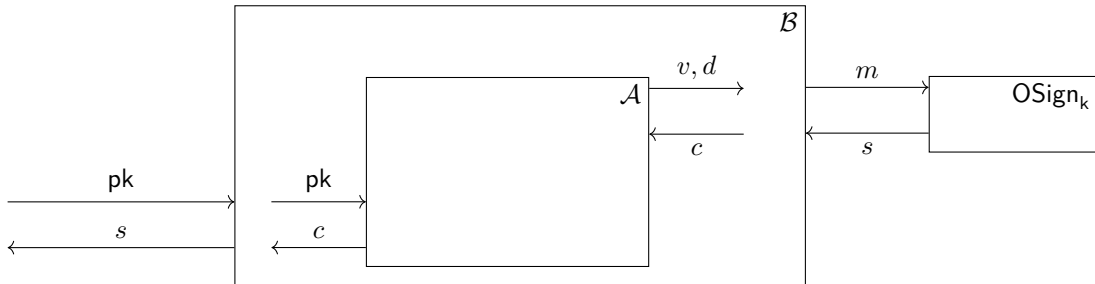


Figure 5.31: Adversary $\mathcal{B}$; capable of winning the EUF-CMA security game, as given in Figure 5.2, given the existence of adversary $\mathcal{A}$ from Figure 5.30. Queries $\mathcal{A}$ makes to the credit generation oracle are passed through to the signing oracle available to $\mathcal{B}$. The forged credit output by $\mathcal{A}$ is passed through as the forged signature output by $\mathcal{B}$.

□

Lemma 5.8.5 (Fair Resource Allocation). Assuming nodes are unable to freely discard debtor identities, in the *Untrusted Node* security model, for DSCS node (i_1, c_1) the computational resources allocated to programs organised by i_1 on the DSCS network c'_1 is proportional to their computational power

$$c'_1 \propto c_1 \quad (5.16)$$

Proof. In order to leverage a disproportionate amount of the network computation a node must circumvent the credit system in some way, as this is the only way to coerce nodes into performing unequal amounts of work. As long as the signature scheme used by DSCS to construct the credit scheme is EUF-CMA secure then by Lemma 5.8.4 the ability of an adversary to forge credits is negligible. As such, a node can not forcibly leverage more compute than they can donate themselves to the network. □

Lemma 5.8.6 (Low Longevity Cost). Assuming that the number of nodes on the network at any given time is independent of the runtime of the network, in the *Untrusted Node* security model, the local storage requirements to actively participate in the DSCS network is independent of the runtime of the network.

Proof. By considering the tokenised credit construction, as defined in Figure 5.20, in the worst case the node need only store a single token per node. Therefore, as before, the network storage requirements are at worst linear in network size $\mathcal{O}(N)$. □

Theorem 5.8.7 (TCM in Untrusted Node Security Model). In the *Untrusted Node* security model DSCS satisfies the properties of a TCM.

Proof. As DSCS is a *Distributed SMUS*, *Decentralised SMUS* and has the *Fair Resource Allocation* and *Low Longevity Costs* properties in the *Untrusted Node* security model it satisfies the conditions of being a TCM. □

5.9 Evaluation

This chapter isolated the requirements of general purpose secure multi-user computation platforms, SMUSs, and a further restricted set of requirements of secure distributed computation platforms, TCMs. These definitions were followed by an exemplar distributed computation protocol, DSCS, which provides a novel secure computation platform for distributed edge devices. It satisfies the properties of a TCM as shown in Theorem 5.8.7 and as such, to the best of the authors knowledge, it is the first fully decentralised secure distributed computation protocol. With the addition of a novel credit scheme, as well as VC and optional FE, DSCS manages to sustain the properties of a TCM even in the face of increasingly malicious adversaries. Unfortunately, not all of the TCM properties are supported by formal proofs, and some are only backed by heuristic arguments. Importantly, the capabilities of DSCS can be weighed against adversarial expectations to include only the additional cryptographic machinery where required. Table 5.3 highlights the major differences between the DSCS protocol and the related work in order to better highlight the niche it has been designed to fill. Where FaaS and conventional multi user operating systems are isolated with security based on superuser isolation, DSCS is distributed with cryptographically backed data security. Where Ethereum and other smart contract platforms are distributed but slow, DSCS is distributed and somewhat fast, due to lack of a resource constrained blockchain backed consensus.

Implementation	Data Confidentiality	Distributed	Low Resource Cost	Speed
Multi User OS (i.e. Linux)	Isolated	No	Yes	Fast
Ethereum Smart Contracts	Public	Yes	No	Slow
FaaS	Isolated	No	Yes	Fast
DSCS	Encrypted	Yes	Yes	Somewhat Fast

Table 5.3: SMUS feature comparison table

First, consider a program comprised of operations over public data. The following arguments about latency operate over abstract computation cost c , and are used to reason about the relative expected latency of programs over abstract DSCS networks. Given a computational task with abstract computational cost of parallelisable sections c_p and computational cost of fully sequential sections c_s , the expected latency for executing task locally is:

$$t_{\text{local}} = c_s + c_p \quad (5.17)$$

For the sake of simplicity the parallelisable sections, c_p , are assumed to be infinitely parallelisable and as such will fully utilise any collaborating nodes. This assumption works for small groups of collaborating nodes, as when working homomorphically often 100 degrees parallelism can be witnessed. This assumption deteriorates at arbitrarily high node counts, but the experimental work in subsequent chapters never reached this ceiling. By delegating the task to a DSCS network of n nodes the expected time to execute becomes:

$$t_{\text{DSCS}} = c_s + \frac{c_p}{n} + o_n n + o_i(c_s + c_p) \quad (5.18)$$

where o_n is the operational cost per-node when engaging with the network, and o_i is the operational cost per-instruction. o_n is incurred when a *Work Group* of peers. o_i represents the average cost to transmit individual instructions across the network within a work fragment. For the sake of simplicity n does not include the *Work Issuing Node*, which is assumed to do no work beyond distributing *Work Fragments* and collecting results.

By considering a hypothetical completely parallel program where every instruction in task c is parallelisable, an upper bound can be obtained for the expected performance of DSCS. This derives a minimum time to execute a DSCS networked computational task, t_{min} :

$$t_{\text{min}} = \frac{c}{n} + o_n n + o_i(c_s + c_p) \quad (5.19)$$

Similarly, a worst-case theoretical bound on performance can be obtained by considering a fully sequential task to determine t_{max} :

$$t_{\text{max}} = c + o_n n + o_i(c_s + c_p) \quad (5.20)$$

Note that this upper limit on time, t_{max} is always worse than the expected time to execute c locally, due to the overheads involved in engaging the network.

These t -values provide bounds on the expected time required to execute a program on the DSCS network.

$$t_{\text{min}} \leq t_{\text{DSCS}} \leq t_{\text{max}} \quad (5.21)$$

$$t_{\text{local}} \leq t_{\text{max}} \quad (5.22)$$

Given a computational task with sufficient parallelisable elements, and low enough operational costs, when

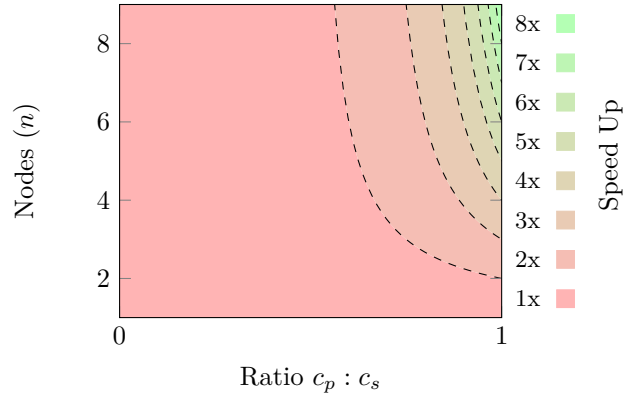


Figure 5.32: A map of the possible speed ups for a given program depending on the ratio of parallel work to sequential work, $c_p : c_s$, and the number of available nodes n . Speed ups are given as a multiplicative factor over executing the target computation with a single *Worker Node*.

engaging with DSCS the expected latency from task inception to obtaining results will be reduced. The latency of some theoretical DSCS task is given in Figure 5.33 for an variable number of nodes.

Given the number of collaborating nodes the speedup potential of a program is capped by the ratio of parallel work to sequential work, $c_p : c_s$. The relationship between $c_p : c_s$, the number of collaborating nodes, and expected speedup is diagrammed in Figure 5.32. Assuming that the parallelism is not saturated, speedups are linear in the number of collaborating nodes if the target computation is completely parallel. However, as Figure 5.32 demonstrates returns diminish quickly as a program has more unparallelisable sections.

An emergent feature of DSCSs design is that it opens edge devices to the possibility of using LFHE evaluation. This allows devices to encrypt data at all times, massively improving the security in the face of device theft and physical tampering. The LFHE schemes used in DSCSs bring trivial parallelism to computational tasks that would otherwise be fully sequential. Evaluating via LFHE adds a linear factor of complexity to an computational task due to a per-operation expansion factor e . The expected latency when evaluating a task homomorphically on the DSCS network is given by:

$$t_{\text{HEDSCS}} = \frac{e(c_p + c_s)}{n} + o_n n + o_i(c_p + c_s) \quad (5.23)$$

As before it is assumed that the degree of parallelism injected by LFHE does not saturate the collaborating nodes. In reality this feature is dependent on the cryptosystem parameters, but in practice it is hard to saturate this configurable limit. A similar restriction involves the integration of VC which, if lifted from Chapter 3, removes any RNS-driven parallelism. Regardless, with very high node counts one should expect to see improvement in computational latency flatten out as the parallelism of the task is saturated by the available worker nodes. The point at which this occurs depends on the LFHE scheme backing DSCS, the parameters it was instantiated with and the choice of VC scheme. Figure 5.34 provides a diagrammatic representation of the expected latency of an LFHE process as the number of nodes is varied. This diagram compares the LFHE DSCS latency, t_{HEDSCS} to local-only plaintext latency, $t_{\text{local}} = c_p + c_s$ and local-only LFHE latency, $t_{\text{HE}} = e(c_p + c_s)$.

Clearly when only considering latency LFHE as it currently exists is not an exciting prospect. The current latency is lower-bounded by the minimum-ciphertext size as dictated by the cryptosystem, parameters, and how ciphertexts decompose into ciphertext fragments. If a ciphertext fragment is an order of magnitude

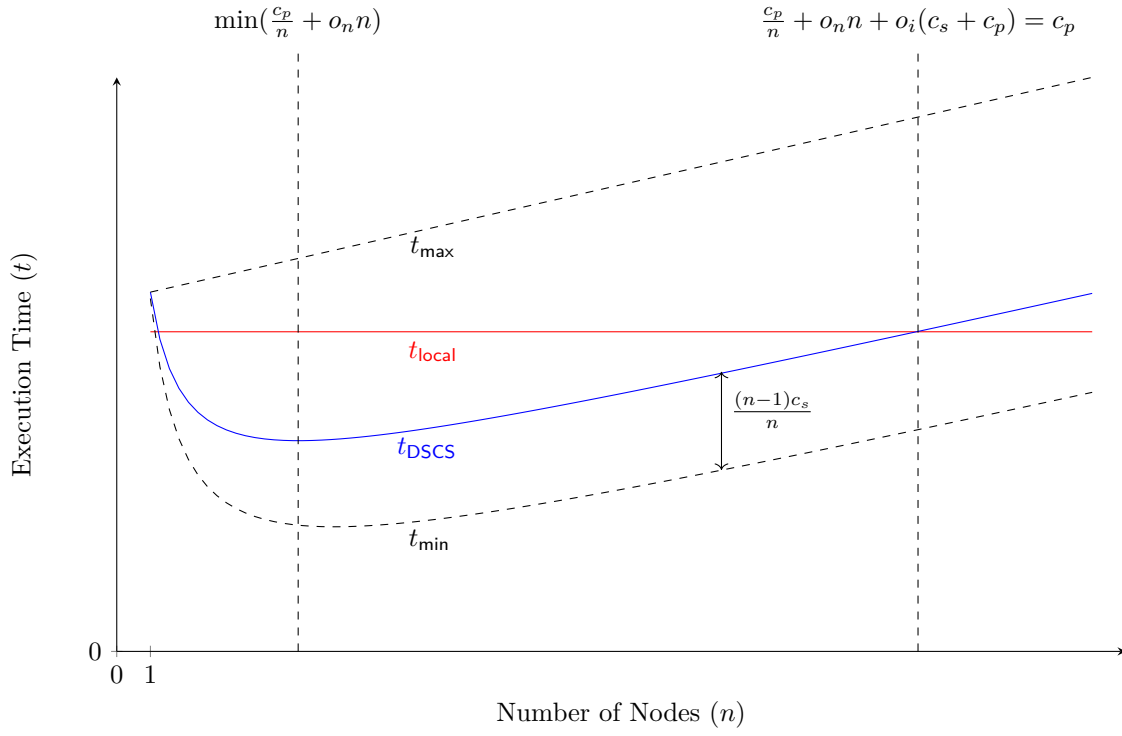
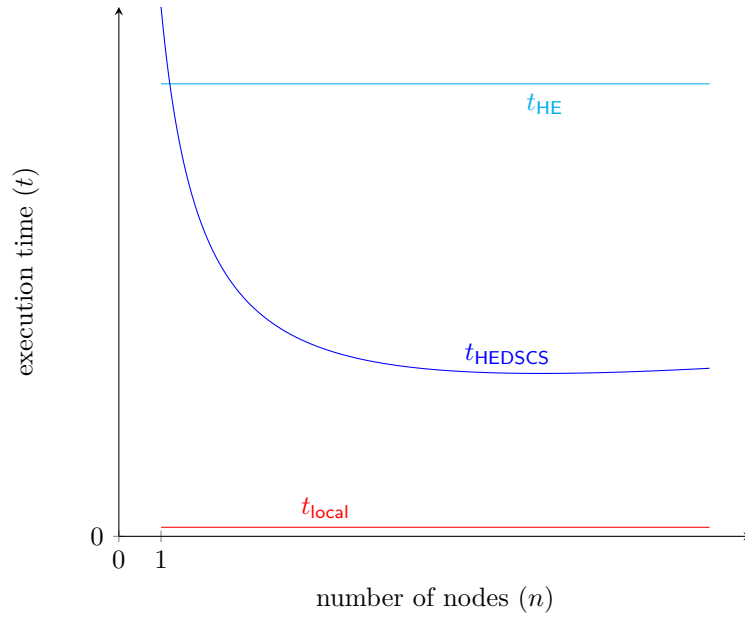


Figure 5.33: The theoretical latency of a computational task delegated to the DSCS network.

Figure 5.34: Theoretical DSCS LFHE ciphertext evaluation latency t_{HEDSCS} as compared with local-only plaintext execution latency t_{local} and local-only LFHE ciphertext evaluation latency t_{HE} , assuming no VC or VC that does not inhibit parallelism.

larger than a plaintext similar inflation can be expected in the latency to process this atomic operand. If the LFHE RNS representation could inject more parallelism into the evaluation process, and reduce the smallest atomic encrypted operand, the program could better saturate a larger group of nodes and the latency would improve. If future work is able to reduce these constraints the latency of programs running on a DSCS network will see improved latency beyond the current scaling limit. Despite the increased computational cost of LFHE, it does provide a remarkable increase in data security. By using DSCS, the burden of LFHE on the individual node is massively reduced. This provides relatively low-latency data processing with security guarantees hardened against even device physical theft and computer forensics.

Nodes pay a compute premium to accelerate programs with a DSCS network. This cost arrives in the per-instruction and per-node overheads, o_i and o_n respectively. The efficiency of non-LFHE DSCS accelerated programs is defined as:

$$\text{Efficiency} = \frac{c_p + c_s}{o_i(c_p + c_s) + o_n n} \quad (5.24)$$

across all collaborators, where o_w is the overhead incurred by an individual *Worker Node* as they process and schedule network communication. The loss of efficiency is defined by any overheads.

Network throughput is dependent on the size, number, and frequency of tasks arriving on the network and is hard to quantify concretely. It is, however, easy to see that given overheads smaller than a large upper limit on efficiency, programs running on DSCS allow for a greater utilisation of the computational capabilities of the network nodes. This in turn will result in both less time spent idle and a reduction in the latency of computational results.

5.10 Further Work

The work presented in this chapter constitutes the first secure decentralised distributed computation protocol, with six broad areas for improvement:

1. Reconciling VC with FE.
2. A more expressive instruction set.
3. Decentralised data storage.
4. Stronger resistance to identity regeneration attacks.
5. A game-theoretic analysis of the accumulation of credits.
6. Empirical experiments with a complete DSCS implementation.

The first of these involves the integration of VC with FE, as first discussed in Subsection 5.8.3. Currently when instantiating DSCS one must chose between verifying the output of delegated work fragments or the fair exchange of credit. To naively combine the two would involve the fair exchange of credit for unverified work, which is then verified after the worker already holds the credit. FE has the implicit notion that both parties obtain the data they were hoping for from the other party. This can be enforced by providing ZKPs that the ciphertexts swapped at the onset of the protocol are valid encryptions of data with an agreed upon property. This property could be “Is the correct output of evaluating work fragment w ”, but in practice it is unclear how this would be implemented without massive computational investment from both parties. This has potential to violate the *Outsourceable* property of VC schemes, which the composition of the two protocols

must maintain. The mechanisms presented must satisfy all the definitions of FE while also satisfying the definitions of VC. As two party FE is an entirely new class of protocol, one realised for the first time in this thesis, it is possible that new two party FE constructions will arrive that better bridge the gap between the VC and FE.

Currently, *Work Fragments* use a reduced instruction set, such that when constructed they have a fixed finite execution time. This restriction was established to make credit value agreement trivial, as the path taken though a given program is fixed. With no branching instruction, there can be no disagreement between interested parties in the number and type of instructions executed, and therefore the requisite compensation due. However, this restriction can be lifted by requiring all incoming results are validated with VC that also validate the path taken through a program. Although this is VC property is not mandated by the VC definitions, it is natural that a mechanism for checking results is also able to provide some guarantees about the path taken through a program.

Some of the data-storage arguments made in relation to DSCS as a SMUS are restrictive. Data is implicitly owned by a node, who sends it to other nodes implicitly granted read-permission to that data. There is no concept of data shared between nodes. If DSCS had this capacity modern computational models would map better to this new computing paradigm. Such a feature could be constructed with the addition of distributed data storage [42], but such work is beyond the scope of this thesis.

As was discussed in Subsection 5.8.4, the current defences against malicious users disregarding identities that have developed a poor reputation are weak heuristic measures at best. Currently nodes are incentivised to discard identities by virtue of the fact that upon generating a fresh identity they must actively engage with their peers to earn credit to “unchoke” themselves. This is a relatively minor amount of work for a sufficiently motivated adversary. Two potential solutions include centralising identity generation, and making identity generation a one-time arbitrarily hard task. The former is suitable for an IoT deployment, where enrolling new devices already requires manual intervention from the device owner, who can act as a certificate authority and centralise identity generation. The latter is more hypothetical and would require further research, but would involve making identity generation cost a large preconfigured amount. Taking inspiration from proof-of-work based consensus [89], this could be achieved by requiring identities that, when processed through a hash function have a prerequisite number of 0s at the beginning of the hash.

The implications of the credit system on the behaviour of nodes is not fully understood. A game theoretic analysis of idealised node behaviour, both honest and malicious, would provide a strong foundation upon which to base further discussion around the effects of incentivising outsourced computations with tokens of equivalent value. There are outstanding questions about the incentives to hoard or rapidly liquidate tokens given certain environmental conditions, and the impact this has on the computational throughput of the network.

There has been little work in this chapter detailing the empirical overhead of computing via DSCS, a DSCS implementation and the implications therein are the focus of the following chapter, Chapter 6. The asymptotic properties of DSCS are desirable but, as has been shown in the past by numerous LFHE constructions, often this is not enough to encourage adoption. An asymptotically linear factor could be an overwhelming multiplicative increase in computational cost, which is a particularly unappealing concept for a computation platform. Optimisation work can be done to build a protocol that satisfies the properties of a TCM with better performance implications for network nodes.

Furthermore, work exploring a partially centralised protocol, such as one with a centralised set up, could yield fruit in the IoT domain. In such a setting devices would be enrolled in the network, and given one and

only one identity to use. This would eliminate identity regeneration attacks and the guarantee of persistent identities would strengthen security arguments relating to auditability.

5.11 Conclusion

This chapter defined two families of computational substrates. The first, an SMUS, which captures the modern understanding of a secure multi-user system. The second, a TCM, which is a subclass of SMUS that is also decentralised, distributed, and approachable for participation from edge devices. This chapter then constructed a candidate TCM, DSCS, in three security models of increasing adversarial strength: *Nosy Nodes*, *Lazy Nodes*, and *Untrusted Nodes*. Extensions were made to DSCS so it was able to maintain the properties of a TCM in each of these security models. In some cases these properties were supported by rigorous proofs, where necessity required other properties were supported by less rigorous arguments.

To the best of the authors knowledge the computational substrate design space encapsulated by the TCM has never been studied before, and DSCS marks the first construction to satisfy the stipulated design goals. There are some minor issues relating to the composition of VC and FE as presented in earlier chapters, but the protocol presented here provides a largely secure-by-default computational substrate for distributed devices.

Chapter 6

DiSCuS: A Partial DSCS Implementation

DiSCuS is a partial implementation of the DSCS protocol, intended to demonstrate the efficacy of a secure, decentralised, and distributed computation platform. It is only a partial implementation of DSCS as defined in the *Nosy Node* security model, and this work was undertaken in parallel to the development of the DSCS protocol in the previous chapter. The construction of this system was intended to validate and inform architectural decisions taken during the development of DSCS. The work contained within this chapter bridges the gap between theory and practice for distributed computation protocols, providing client-side algorithms, and choosing concrete cryptographic building blocks. These heuristic algorithms are provided alongside quantitative measurements of DSCS performance, validating the claims made in the previous chapter. As with the chapter before this work is summarised in a yet-unpublished paper titled “DSCS: Towards An Extensible Secure Decentralised Distributed Computation Protocol”. 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 work highlights a slew of low hanging fruit for further research in this domain, as well as guidance to avoid easy pitfalls.

6.1 Introduction

In the previous chapter the expected execution latency of programs executed on a distributed computational protocol was outlined in a sequence of equations, ultimately Equation 5.18. This equation, and related expected latency equations, had a number of unknown variables that would directly impact the ability of DSCS to deliver on the promise of improved execution latency. Unknown quantities such as the per-instruction overhead o_i could overwhelm any benefits in distributing a program across a network of collaborating nodes. This chapter intends to explore these unknown variables, and identify architectural improvements and considerations to be made in an actualised DSCS deployment.

The expected latency of a DSCS accelerated program was previous defined in Equation 5.18:

$$t_{\text{DSCS}} = c_s + \frac{c_p}{n} + o_n n + o_i(c_s + c_p)$$

and for HE backed evaluation, Equation 5.23:

$$t_{\text{HEDSCS}} = \frac{e(c_p + c_s)}{n} + o_n n + o_i(c_p + c_s)$$

Variables c_s and c_p are an abstract measure of the computational cost of the target program, n is the number of collaborating nodes, and e is a fixed expansion factor for a chosen underlying HE scheme. The remaining variables, o_n and o_i , are not only unknown but also largely dictate the utility of DSCS as a computation platform for a target program. Any network latency is assumed to be linear in the number of instructions included in a *Work Fragment*, and as such this is included in the per-instruction overhead, o_i . This abstraction does not take into account the specific network topology nor the corresponding inter-node communication capacity.

This simplified computational model does not take into a myriad of factors which would affect the efficacy of a DSCS deployment including:

- Variance in computational capacity between nodes.
- Network topology.
- Inter-node communication bandwidth.
- Inter-node communication latency.
- Probabilistic model of the computational tasks of network peers.

These factors have been largely ignored to simplify the model. The experiments in this chapter control for these ignored variables and execute the benchmarks under idealised conditions to measure performance in the optimistic case.

The performance of DSCS is also dependent on the choice of concrete cryptographic components previously specified as abstract black boxes. Within this chapter contemporary constructions are compared against each other and their relative merits considered. A concrete choice must be made for an RNS-parallelisable IND-CPA secure LFHE scheme. Picking an LFHE scheme is not as easy. The computational effort spent in LFHE functions within DSCS means the LFHE performance has a large impact on the overall performance of DSCS. Due to the requirement that the LFHE is parallelisable, the potential candidate LFHE schemes are the schemes in which evaluation instructions easily translate to arithmetic operations over rings. Properties of various LFHE schemes were discussed in more detail in Subsection 5.3.3. BFV was chosen as the LFHE scheme as it has strong performance characteristics relative to related works, and is a popular scheme for implementation, and relative implementation simplicity. Homomorphic addition directly translates to a pair of additions over a ring, and multiplication can be manipulated with pre- and post- processing to produce a similar result.

Naturally there will be some effort required to bridge the gap from the theory presented in the previous chapter and an implemented protocol. This chapter intends to make headway in this direction addressing engineering challenges as they appear in the implementation process. Furthermore, this chapter is used to validate the claims made in the previous chapter and obtain quantitative performance results for a DSCS implementation.

Only the simplest incarnation of DSCS is implemented, and only with a further reduced instruction set for *Work Fragments*. As such the security model of choice is that of the *Nosy Node* security model. This was deemed sufficient to inform designed decisions of DSCS. Some instructions are unimplemented as their

genesis came out of insights gained due to the outcomes of the implementation work. Implementing the extended protocols presented in the previous chapter would yield an interesting direction for further research engineering work. Undoubtedly this would uncover further interesting engineering challenges, beyond those captured in this chapter.

The reduced instruction set implemented in DiSCuS is a direct result of the results gathered here feeding back into the design of DSCS. Before implementation work was completed the LOOP instruction was not included in DSCS, it was only after observing the real-world factors limiting execution latency that the inclusion of this addition instruction was mandated. Unfortunately there was not enough time for this to be propagated to DiSCuS.

Beyond exploring the real-world costs associated with a DSCS deployment, this chapter indicates four client-side algorithms that require specification by a DSCS implementer. These algorithms include:

1. Work group prioritisation.
2. Anti-credit hoarding heuristics.
3. Work group saturation limits.
4. Programmatic cryptosystem parameter generation.

In this work, in-keeping with the choice of security model, the simplest conceivable algorithm was chosen for each required algorithm. Namely, work groups are joined greedily, worker nodes are added to work groups greedily, and, as credit is not defined as part of this security model, the anti-hoarding heuristics can be ignored.

6.1.1 Contributions

The contributions of this chapter are:

- Details of a DSCS reference implementation, the engineering challenges, and other factors for implementation consideration.
- Comparison of candidate underlying cryptographic constructions, and recommended choices.
- Details of the close integration of DSCS and BFV, the LFHE of choice for DiSCuS.
- Example local-only algorithms and heuristics required for a DSCS client application.
- Quantitative performance measurements for DSCS accelerated program execution.
- Identifying low hanging fruit improvements for further work.
- Signposting avoidable pitfalls in distributed computation protocol design and implementation.

6.2 Background

This section details required background for this chapter. As DiSCuS is an implementation of a subset of the DSCS protocol the relevant definitions from DSCS are restated from Chapter 5. Following from this the LFHE cryptosystem of choice, BFV, is specified with reasons given for the selection over other HE cryptosystems. This is followed by a common acceleration technique for cryptosystems over rings, the representation of

ciphertexts via DCRT, which serves as a logical extension of residue number systems, as originally detailed in Chapter 2, to the domain of high order polynomials.

6.2.1 Notation

The LFHE cryptosystem used in this work, BFV, is constructed on high order polynomial rings. For large integer N the polynomial ring $R_N = \mathbb{Z}_N/(1 + x^i)$ is defined, which has large order i and integer coefficients taken modulo N . As in previous chapters, N is often taken as the product of a set of fixed bit-width prime numbers, P_N , such that $N = \prod_{p \in P_N} p$. Often the polynomial rings R_M and R_{NM} are also referenced, these are constructed with M , another large integer product of fixed precision primes $M = \prod_{p \in P_M} p$ such that M is much larger than N , and M and N are co-prime. R_{NM} is the ring defined by $\mathbb{Z}_{NM}/(1 + x^i)$, the ring of polynomials with coefficients taken modulo NM . When operating within a ring, x^{-1} is the multiplicative inverse of x , if it exists.

Where it would otherwise be unclear, a reduction of a polynomial p to a ring R_N is specified with the notation $[p]_N$. $\lfloor x \rfloor$ represents rounding x down to the nearest smaller integer, and $\lceil x \rceil$ signifies rounding x either up or down to the nearest integer. In order to maintain parity with background work on LFHE schemes $\mathcal{X}$ is used as a discrete Gaussian distribution over a ring R .

In order to make translation to implementation more concrete sets, usually denoted by an unreserved capital letter, are treated as vectors and as such are assumed to be ordered. Individual elements in set S at index i can be accessed via the notation $S[i]$. Matrices have a similar index notation, where the value in the i th column and the j th row of matrix m can be accessed with $m[i, j]$.

6.2.2 DSCS

As a reminder from Chapter 5, DSCS is a decentralised distributed computation protocol designed to accelerate the execution of programs by distributing parallel partial programs across a swarm of collaborators. The protocol is tightly integrated with LFHE primitives and this allows the newly introduced trivial parallelism to be exploited to accelerate LFHE-backed programs. DSCS is defined for a variety of security models, the simplest of which is the *Nosy Nodes* security model, and this is the chosen DSCS variation for this implementation work.

In the *Nosy Nodes* security model DSCS nodes present a relatively simple public interface, with heuristic internal behaviours provided in the state machine in Figure 5.14. If the DSCS public behaviours are reinterpreted as a suite of functions, a DSCS node must implement the following public-facing functions:

- **Advertising Query:** $\top$ or $\perp \leftarrow \text{Ad}()$, returns $\top$ if the node is currently accepting nodes into a work group, $\perp$ otherwise.
- **Join Work Group:** $\top$ or $\perp \leftarrow \text{Join}(i)$, register to join the work group organised by this node with identity i . $\top$ on acceptance into the *Work Group*, $\perp$ otherwise.
- **Leave Work Group:** $\top \leftarrow \text{Leave}(i)$, leave work group organised by this node.
- **Submit Work Fragment** $\top \leftarrow \text{Submit}(o)$ submit the result of an evaluated work fragment o .

These functions have a one-to-one correspondence to the states given in the simplified state machine defined in Figure 5.13, and duplicated in Figure 6.1.

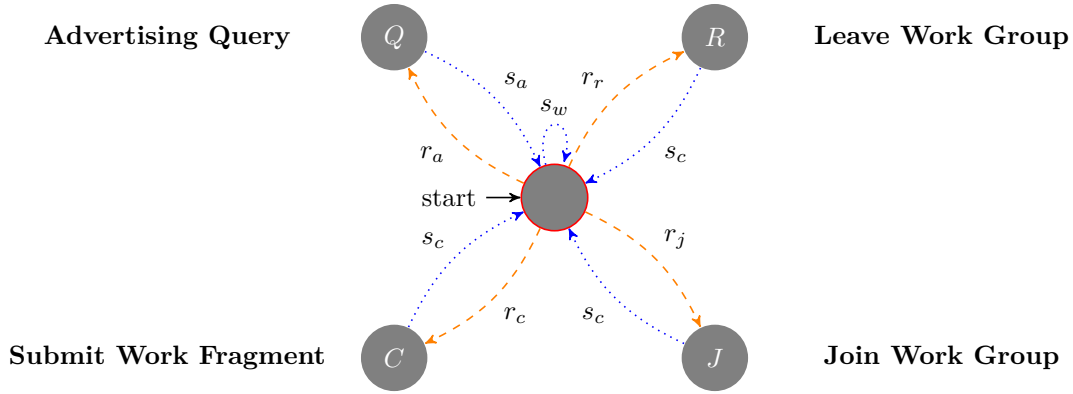


Figure 6.1: Public DSCS *Work-Issuing Node* interface in the *Nosy Nodes* threat model, duplicated from Figure 5.13, annotated with the implementation target functions.

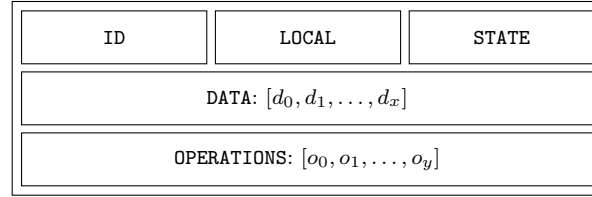


Figure 6.2: DSCS Work Fragment, duplicated from Figure 5.6.

Work Fragments are the objects used to capture discreet units of outsourced computation. The structure of a *Work Fragments* is reiterated in Figure 6.2, and the expected structure of the object containing results is given in Figure 6.3. Both definitions are duplicated from Chapter 5.

6.2.3 BFV

An implementation of DSCS in all of the defined threat models requires selecting concrete constructions for the various cryptographic building blocks, including VC, signature schemes, and FE. Chapter 5 provided some guidance for choosing these in Subsection 5.3.3. However, in this chapter the only choices that must be made to satisfy the minimal target set of DSCS behaviours, is selecting an LFHE scheme. As a reminder from Subsection 5.3.3, LFHE schemes are encryption schemes that allow addition and multiplication of encrypted data without a decryption key, given that the number of execution steps is known in advance.

BFV [52] is a popular LFHE scheme based on the RLWE problem [84], which itself is an extensions of the well-studied Learning With Errors (LWE) problem [98] from rings of integers to polynomial rings. The LWE and RLWE assumptions are widely accepted and shared by some of the candidate post-quantum cryptosystems currently undergoing NIST standardisation [5]. The definition of BFV is given as definitions to the standard LFHE functions (as specified in Subsection 5.3.3) in Figure 6.4. Plaintexts are elements of the polynomial ring R_t , and ciphertexts consist of pairs of elements from $R_N^2 = R_N \times R_N$, made up of pairs

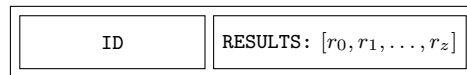


Figure 6.3: Results object structure, duplicated from Subfigure 5.7c

of polynomials.

Notably, the definitions for BFV deviate somewhat from the standard LFHE function definitions by providing an additional function `Relin`: a relinearisation function used to reduce the invalid ciphertexts produced by the vector multiplication in `EvalMul` from R_N^3 to the valid ciphertext space of R_N^2 . The relinearisation function requires an additional public key `rlk`, in effect this key is an obfuscated version of the decryption key, and allows an untrusted remote party to perform the relinearisation step. In practice the relinearisation key is stored within a structured object alongside the encryption public key. This functionality can be less elegantly expressed without deviating from the standard LFHE function definitions by folding the functionality of `Relin` into `EvalMul`, and careful key management.

BFV is a LFHE due to a per-ciphertext error value that accumulates as operations are executed. When this error overflows a configurable bound ciphertexts become un-decryptable. The per-ciphertext error grows linearly with ciphertext additions, and multiplicatively with ciphertext multiplications. In programs with a mixed sequence of operations often it is suitable to configure BFV to support the expected number of multiplication operations and ignore the addition operations as insignificant contributors to ciphertext error. Due to various levels of indirection within the soundness proofs of lattice based cryptosystems, parameters are often chosen based on heuristic values to improve performance at the cost of a marginal probability of ciphertexts becoming un-decryptable.

Another popular LFHE scheme is LTV [81]. Experiments early in this work were done using the LTV encryption scheme as, at the time, it was still a popular choice for LFHE implementation. This popularity was mostly driven by relative implementation simplicity, with `EvalAdd` and `EvalMul` functions that directly translated to polynomial addition and polynomial multiplication respectively. LTV is based on an extension to the NTRU lattice-based encryption scheme [72], an extension which more recent work has shown makes the scheme vulnerable to attack [6]. This vulnerability has caused LTV to fall out of fashion, with BFV becoming the more popular implementation target, as BFV is only slightly more complex to implement than LTV with no known security flaws at the time of writing. The requirement to substitute BFV for LTV after a discovered vulnerability [6] highlights the value in maintaining a high degree of abstraction when building complex cryptographic constructions from existing cryptographic machinery.

When generating parameters for BFV, although it is possible to generate for N such that errors during decryption are impossible, it is often sufficient to set N much lower than this upper bound. Experimental evidence suggest for most uses this value can be optimistically generated given the plaintext modulus, a target computational depth, and a security parameter [70]. Lattice-based cryptosystems are infamously hard to select parameters for due to loose “tightness” in the various reductions to foundational lattice hard problems [36]. BFV is no outlier to this problem, and parameter selection is informed by experimental failure rates.

6.2.4 Double CRT Ciphertext Representation

As previously described, BFV ciphertexts consist of a pair of high order polynomials each in R_N for some large coefficient modulus N . N is often taken as a composite of prime factors in order to make operations compatible with an RNS acceleration technique. When N is set to the product over a set of some relatively small unique prime numbers P_N , ciphertext polynomials can be manipulated when encoded as an RNS over N ’s prime factors, P_N . By doing this, the large ciphertext polynomials are each encoded as $|P_N|$ independent smaller polynomials via a mechanism known as a DCRT [85]. DCRT is the application of RNS techniques from integers to polynomials. In the literature the terms RNS, DCRT and Number Theoretic Transform (NTT)

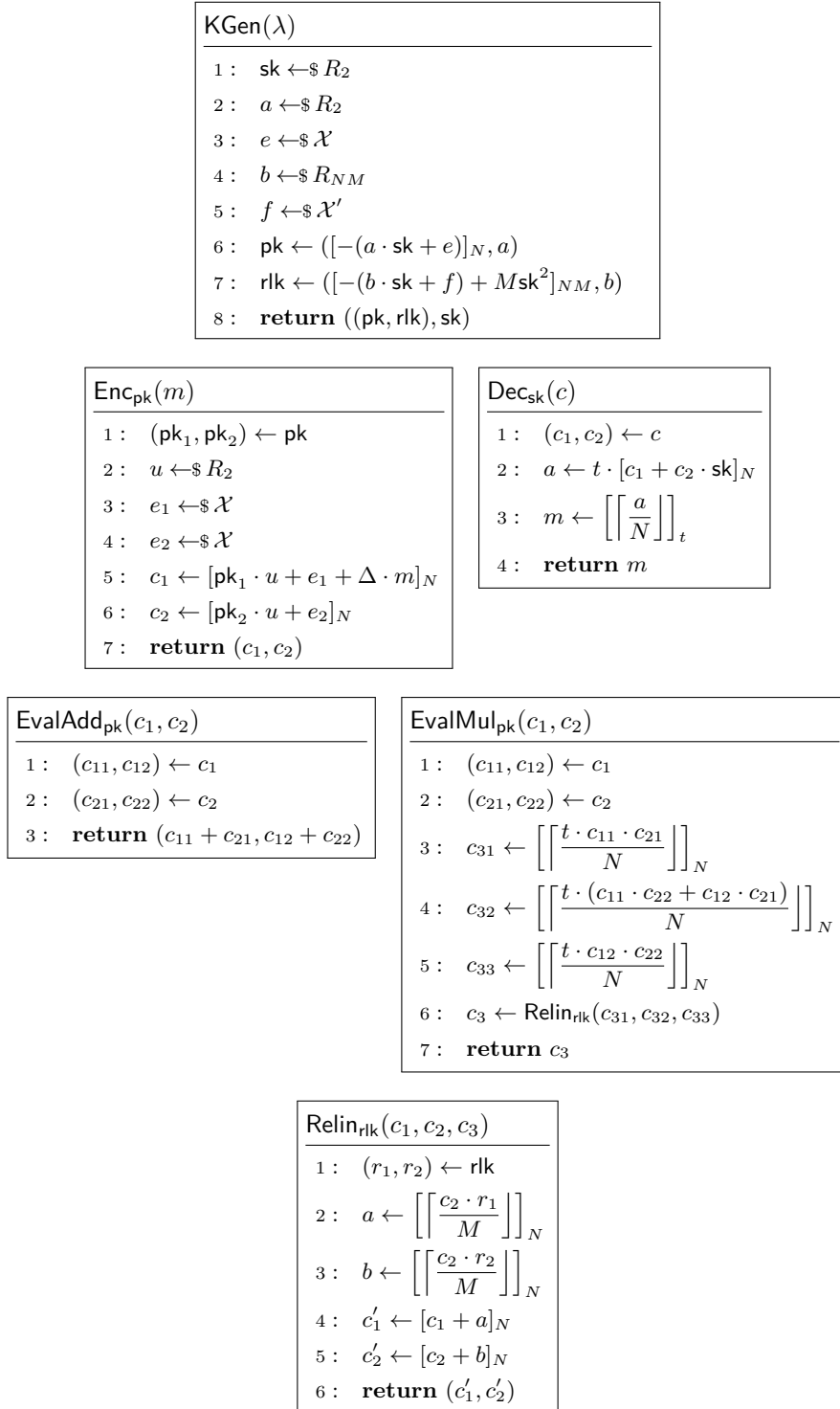


Figure 6.4: BFV cryptosystem function definitions adapted from [52]. Where N is the coefficient modulus for the space of ciphertexts, t is the coefficient modulus for the space of plaintexts, M is an expansion factor for the relinearisation procedure, and $\Delta = \lfloor N/t \rfloor$.

are used interchangeably to refer to the application of RNS acceleration techniques to polynomials.

The conventionally encoded large BFV ciphertext polynomials are indexed by $b \in \{0, 1\}$. Each polynomial is represented as a vector containing the ordered set of polynomial coefficients, $c_b = \{c_{b,0}, \dots, c_{b,i-1}\}$, where i is the ring order and ciphertext modulus is $N = \prod_{p \in P_N} p$.

During the DCRT each of the one dimensional vectors, c_b , are decomposed into a matrix of values c'_b , where each row represents the RNS decomposition of a single coefficient. Each column in this matrix therefore contains each of the coefficients of c_b reduced modulo a single element $p \in P_N$. The matrix constructed row-by-row by taking the vector of coefficients of c_b and reducing each coefficient modulo the next element of P_N .

$$c'_b \leftarrow \begin{bmatrix} c_{b,0} \bmod P_N[0] & \cdots & c_{b,0} \bmod P_N[a-1] \\ \vdots & \ddots & \vdots \\ c_{b,i-1} \bmod P_N[0] & \cdots & c_{b,i-1} \bmod P_N[a-1] \end{bmatrix} \quad (6.1)$$

Operations on c_b are translated into $|P_N|$ identical independent operations each on a different column in c'_b . Before encoding the ciphertext with DCRT evaluating an arithmetic operation between two ciphertexts required access to both large polynomials of both ciphertexts. Now with DCRT representation, only a single column for each component of each ciphertext needs to be accessed at a time. As a result, remote nodes can now perform small parts of a complete HE arithmetic operation. These partial evaluations can be collected, the matrix columns combined into a ciphertext matrix pair, which can then be reconstituted into the pair of vectors representing the result encoded in the original ciphertext polynomial representation. This original representation can then be decrypted as before to obtain the plaintext result.

The process to reverse DCRT is straight forward:

$$\hat{N} \leftarrow \left\{ \left(\frac{N}{P_N[0]} \right)^{-1}, \dots, \left(\frac{N}{P_N[a-1]} \right)^{-1} \right\} \quad (6.2)$$

$$c_b \leftarrow \left\{ \sum_{j=0}^a \hat{N}[j] \cdot c'_b[0, j] \bmod N, \dots, \sum_{j=0}^a \hat{N}[j] \cdot c'_b[i, j] \bmod N \right\} \quad (6.3)$$

An easier way of considering the reconstruction of c_b is that each of its component coefficients are each rebuilt from a single row in the matrix c_b . The set $\hat{N}$ can be precalculated when the protocol parameters are selected.

In order to perform a calculation with a ciphertext in an RNS representation each j th column of c'_b is treated as an independent polynomial in ring $R_{P_N[j]}$. Addition and multiplication operations within the ring R_N are converted to $|P_N|$ independent addition or multiplication operations within the rings $\{R_p : p \in P_N\}$. With the parallelism DCRT introduces, BFV ciphertext evaluation becomes trivially $|P_N|$ -way parallel and has smaller fixed precision operands. If the cryptosystem parameters warrant a larger ciphertext coefficient modulus more parallelism can be added to the system and relatively small fixed precision arithmetic can still be used for homomorphic evaluation. The amount of parallelism introduced is related to the expected maximum depth of the LFHE computation, as this is the value that informs a choice of N and $|P_N|$. Interestingly, as BFV ciphertext addition already has two degrees of parallelism, by using DCRT, ciphertext addition can be further decomposed into a total of $2|P_N|$ independent operations.

Due to a number of non-ring operations used during BFV multiplication (a scaling procedure, and some operations in a larger ring R_{NM}), parallelism via DCRT is less trivial than with other related cryptosystems, such as LTV. There are mechanisms for performing these non-ring operations on individual DCRT compo-

nents [70] but they include sequential steps which come before and after the massively parallel DCRT ring multiplications. In order to trivially translate a LFHE cryptosystem to DCRT ciphertext addition and multiplication must consist exclusively of ring operations. DiSCuS implements BFV with the DCRT multiplication extensions specified in [70].

6.3 DiSCuS

DiSCuS is a distributed program execution framework designed to implement a subset of DSCS features in order to validate claims made about performance and inform further protocol development. It is written in Rust using peer-to-peer primitives from the `libp2p` distributed application library. The DSCS requirement to support operating on partial LFHE ciphertext fragments required implementing the BFV cryptosystem natively in Rust in order to deeply integrate DiSCuS with the cryptosystem primitives and properly expose the independent fragments of BFV ciphertexts expressed in DCRT form. The DiSCuS code base is publicly available on GitLab¹.

DiSCuS implements a subset of DSCS in the *Nosy Node* security model. This subset of behaviours was the complete set of behaviours for an early development version of DSCS in the *Nosy Node* security model, which was later extended based on shortcomings discovered during the development of DiSCuS. The subset of behaviours targeted includes all *Nosy Node* security model functionality except for native support for the `LOOP` instruction in *Work Fragments*. The ability to compress looping structures was identified within this work as essential for the efficient operation of a distributed computation framework.

Beyond the targeted DSCS functionality, DiSCuS aims to provide a suite of tools to more easily interact with a DSCS network. To this end DiSCuS also supports:

- Additional private client-side operations.
- Programmatic cryptosystem parameter selection.
- Processing user-defined programs into *Work Fragments* at runtime.
- Reading and writing DSCS programs from disk.

The additional client-side operations supported by the interpreter include:

- `ENC`: convert an integer to a ciphertext.
- `DEC`: convert a ciphertext to an integer.
- `IF/WHILE`: control flow statements.

These allow writing complex programs which do not strictly fit the DSCS protocol, but instead allow DiSCuS programs to dynamically form DSCS-compatible *Work Fragments* during execution.

Cryptosystem parameters are generated depending on the required execution depth derived from the program, plaintext precision as defined by the programmer, a minimum level of security defined globally in DiSCuS, and heuristics gleaned from experimentation. Processing user programs allows the conversion of instructions into their homomorphic equivalents where necessary, including dynamically adding instructions for encrypting and decrypting as needed. Programs are written as lists of `GlobalInstruction` type which contain all the information required for execution. The programs are executed instruction-by-instruction

¹<https://gitlab.com/AlexDaltonPhD/discus>

until there are no remaining unexecuted instructions. The final step in preprocessing, usually executed immediately before execution begins, is to divide a program into chunks. This decision can be made based on a number of environmental factors, such as network strength, or program parallelism, but for the sake of implementation simplicity DiSCuS divides the instructions evenly between any *Worker Nodes* in the *Work Group*.

When instructions are represented as **GlobalInstruction** they are completely independent of each other and contain copies of any operands they require, and referenced to yet-to-be-created operands when they have unresolved dependencies on unevaluated instructions. During the partitioning of a program into *Work Fragments* the **GlobalInstructions** are converted into **LocalInstructions**, which involves creating a shared data pool for all operations within the *Work Fragment*. **LocalInstructions** need to only contain a reference to the location of any operands within this local pool of data.

The definitions in this subsection, taken from Chapter 5, are discussed in further detail in the following sections. Specific care is given to address the implementation concerns and intricacies where appropriate.

6.3.1 Mapping Conventional Arithmetic onto Homomorphic Evaluation

Unfortunately conventional arithmetic does not cleanly map onto the arithmetic operations provided by HE evaluation. Plaintexts in most HE cryptosystems are high order polynomials, and the arithmetic supported by HE are polynomial arithmetic operations. For applications already easily expressed as polynomial arithmetic operations converting to HE can be done easily, however naively converting between conventional arithmetic and HE arithmetic requires putting a single value, v , in the plaintext polynomial and padding the polynomial with zero coefficients.

$$m \leftarrow v + 0x + 0x^2 + \dots + 0x^{(i-1)} \quad (6.4)$$

Conventional arithmetic addition operations can be vectorised by inserting individual operands into each polynomial coefficient of the HE plaintext.

$$m \leftarrow v_0 + v_1x + v_2x^2 + \dots + v_ix^{i-1} \quad (6.5)$$

Any lattice-based HE addition operations translate into a vectorised addition over the polynomial operands.

Unfortunately homomorphic multiplication, by virtue of its roots in polynomial multiplication, doesn't cleanly map onto vectorised multiplication. To gain maximum utility from homomorphic arithmetic requires manually sculpting plaintexts and handcrafting a sequence of homomorphic operations to a specific use-case. The alternative involves naively padding plaintexts and wasting a great deal of the computational potential of HE arithmetic. Compilers capable of automating this analysis could allow for the effort of maximising the utility of HE to be automated.

6.3.2 Algorithms

Part of the effort involved in implementing DSCS involves developing implementation-specific algorithms intentionally omitted from the DSCS definitions. These omissions allow DSCS to be more concisely defined, with only the public-facing components and some security-centric private heuristics requiring definitions. The client-side algorithms that require definitions are:

1. Intricacies involved in parallelising the evaluation operations of the LFHE of choice and constructing *Work Fragments*.

```

METADATA { ... }
PROGRAM {
    ENV1 { .. }
    DISTRIBUTED { .. }
    ENV1 { .. }
}

```

Figure 6.5: High-level DiSCuS program structure.

2. Programmatic cryptosystem parameter selections.
3. *Work Group* prioritisation.
4. *Worker Node* organisation.

Where possible DiSCuS opts for the simplest possible algorithm.

The first problem encountered in the construction of DiSCuS which is unaddressed in the definition of DSCS pertains to specific implementation complexities with BFV ciphertext multiplication and DCRT representation of ciphertexts. Where BFV DCRT ciphertext addition is a simple sequence of vector addition operations in the ciphertext rings, unfortunately BFV DCRT multiplication is not as easy to parallelise. Multiplication requires sequential steps which come before and after the massively parallel standard ring multiplication, the mechanics of this are detailed in [70].

A more nuanced problem involves the careful selection of cryptosystem parameters in order to achieve a required computational depth and a required degree of security. To this end DiSCuS includes sensible defaults for cryptosystem parameters, with some parameters dynamically based on the computational depth of a target calculation, as previously described in Subsection 6.2.3. Programmers creating tasks to be executed with the DiSCuS programming framework define a desired level of plaintext precision, expressed in terms of a fixed modulus and a maximum safe computational depth - these parameters combined with the sensible defaults produce a secure instantiation of BFV optimised for a specific computation. As future work it would be relatively straight forward to extend DiSCuS to extract the BFV computational depth from a provided program.

A DSCS implementer must also consider the process by which a node prioritises the *Work Groups* of their peers. DiSCuS is designed for use in closed trusted systems and as such each node simply joins any *Work Groups* available. In fact, the mDNS discovery process involves the implicit assignment of any discovered peers to a local perpetual *Work Group*. This can be refined for larger and untrusted networks with further game theoretic work.

Furthermore, clients have the option of implementing logic to prioritise certain nodes with **READY** *Work Fragments* depending on prior behaviour and performance. Currently DiSCuS distributes packets in a round robin style where idle nodes are stored in a double ended queue. Work is assigned in a first-in-first-out manner to nodes in this queue.

6.3.3 Programs

DiSCuS operates on programs written in a domain specific language. Data is tagged with a security level (either secure or public) and instructions mutate the data. Alternatively, users can create programs in a more verbose assembly language style by writing **GlobalInstructions** directly. The benchmark programs written for DiSCuS were created in this way.

A high-level example of the structure of a DiSCuS program is provided in Figure 6.5. Programs written in the domain specific language begin with a **METADATA** environment, where configurable elements are set, such as the desired level of precision, or the maximum expected computational depth. This is followed by a **PROGRAM** environment, in which a sequence of programming environments are defined. Each environment includes code which is executed sequentially, and each environment corresponds to an identity on the network, where the code is executed. There is a reserved environment name **DISTRIBUTED**, this corresponds to sections of code that are automatically parallelised and dynamically distributed across the network. Figure 6.6 shows the workflow through which DiSCuS adaptively converts the program written in the DiSCuS DSL into *Work Fragments*, encodes data dependencies, sends ready *Work Fragments* to participating nodes, collects submitted results and resolves pending data dependencies.

As an example, Figure 6.7 is a program defining a series of two integer additions, the first operating on public data, the second on private data - and so will be automatically translated into their homomorphically encrypted corresponding operations before distributing to the network.

Note that in the worked example the private variables **a** and **b** are not explicitly tagged as such. This is because the language assumes that all data is private unless it is tagged otherwise. DiSCuS provides a standard library of utility functions, such as **getint** and **print** to make interacting with running programs easier. Code fragments are always enclosed within an execution environment. These can be named anything and the intention is that in the future users of the system can use these environment definitions to provided specific functionality - such as voting schemes, or two player games where the processing is handled by the DSCS network. Currently any environment not labelled **DISTRIBUTED** is executed locally, and the **DISTRIBUTED** code fragments are processed on the DSCS network.

When the user launches a program, each code environment is executed in sequence. Code fragments not within a **DISTRIBUTED** environment are interpreted line-by-line on the user's local system. The code fragments delimited by the **DISTRIBUTED** environment are converted into a sequence of **GlobalInstructions** as an intermediate format, and then combined into *Work Fragments* containing **LocalInstructions**. The first step in this process, converting the **DISTRIBUTED** code into **GlobalInstructions** is shown in Figure 6.8.

After **GlobalInstructions** are constructed, the homomorphic operations are parallelised and their operands are converted into DCRT form. This preprocessing is shown in Figure 6.9.

Depending on environmental factors it may be desirable to assemble *Work Fragments* to maximise any number of different characteristics, such as to fit the number of *Work Fragments* to the *Work Group* size to minimise latency, or reduce the number of *Work Fragments* to minimise inter-*Work Fragment* data dependencies. In the worked toy example, the DiSCuS framework may try to saturate a *Work Group* of two nodes by constructing the *Work Fragments* described in Figure 6.10.

These *Work Fragments* are disseminated to the *Work Group*, and the collaborating nodes evaluate the fragments, returning the results. In the worked example the structure of this data is given in Figure 6.11.

6.3.4 Interpreter

Bundled with the DiSCuS framework is a DSCS interpreter. This is used when evaluating on *Work Fragments* from peers and also for deterministically executing programs expressed as *GlobalInstructions* operation-by-operation for development purposes. The interpreter is a simple state machine, where the current state is a combination of the program data, program instructions, and a current program counter. The current instruction is used to mutate the program data, data dependencies are updated, and then the program counter is incremented.

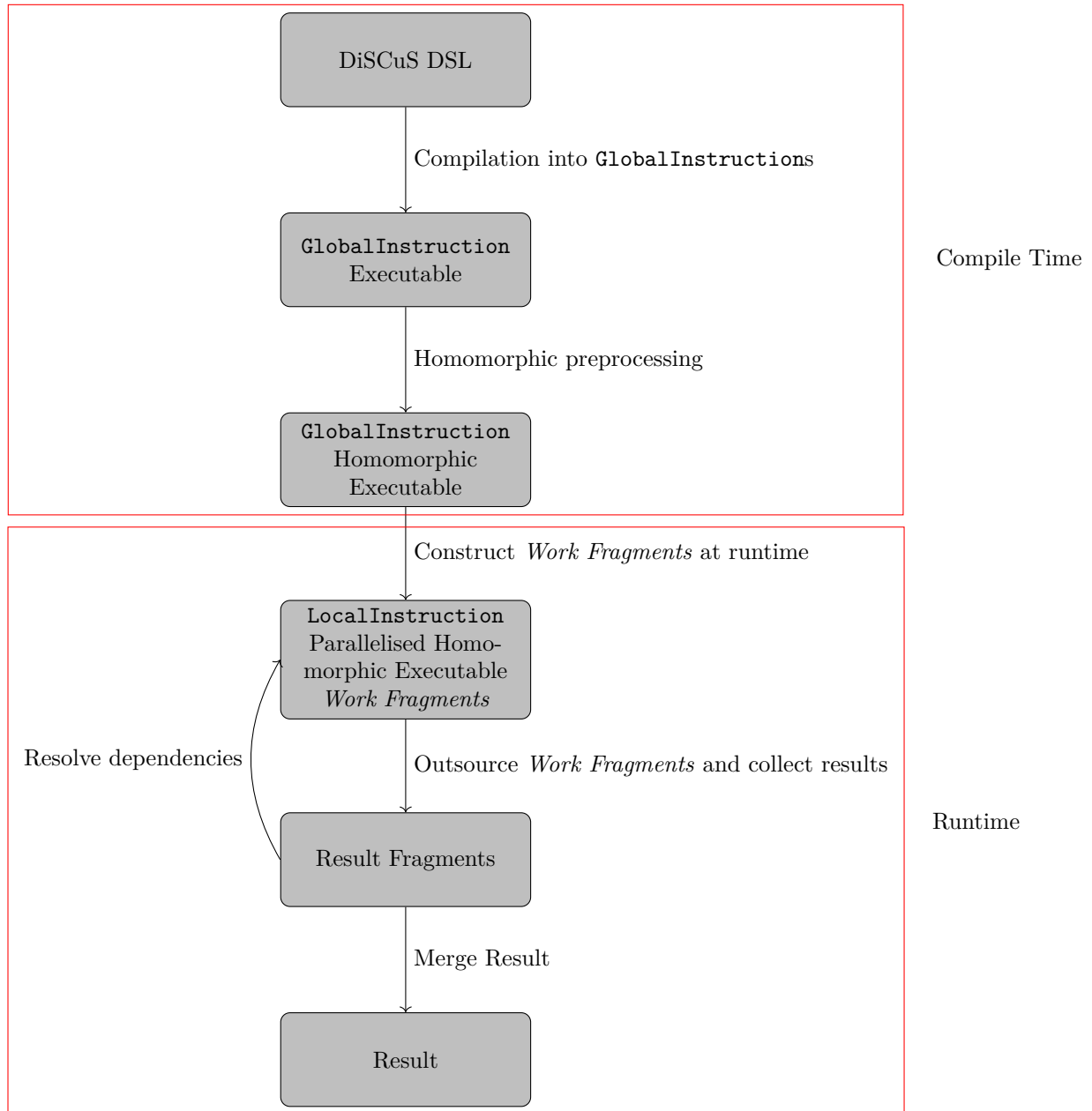


Figure 6.6: DiSCuS Program Processing

```

METADATA {
    DEPTH = 2;
    PRECISION = 8;
    NAME = "Two Additions";
}
PROGRAM {
    ALICE {
        public: a = getint();
        public: b = getint();
        c = getint();
        d = getint();
    }

    DISTRIBUTED {
        sum1 = a + b;
        sum2 = c + d;
    }

    ALICE {
        print("Sum of a + b =");
        print(sum1);
        print("Sum of c + d =");
        print(sum2);
    }
}

```

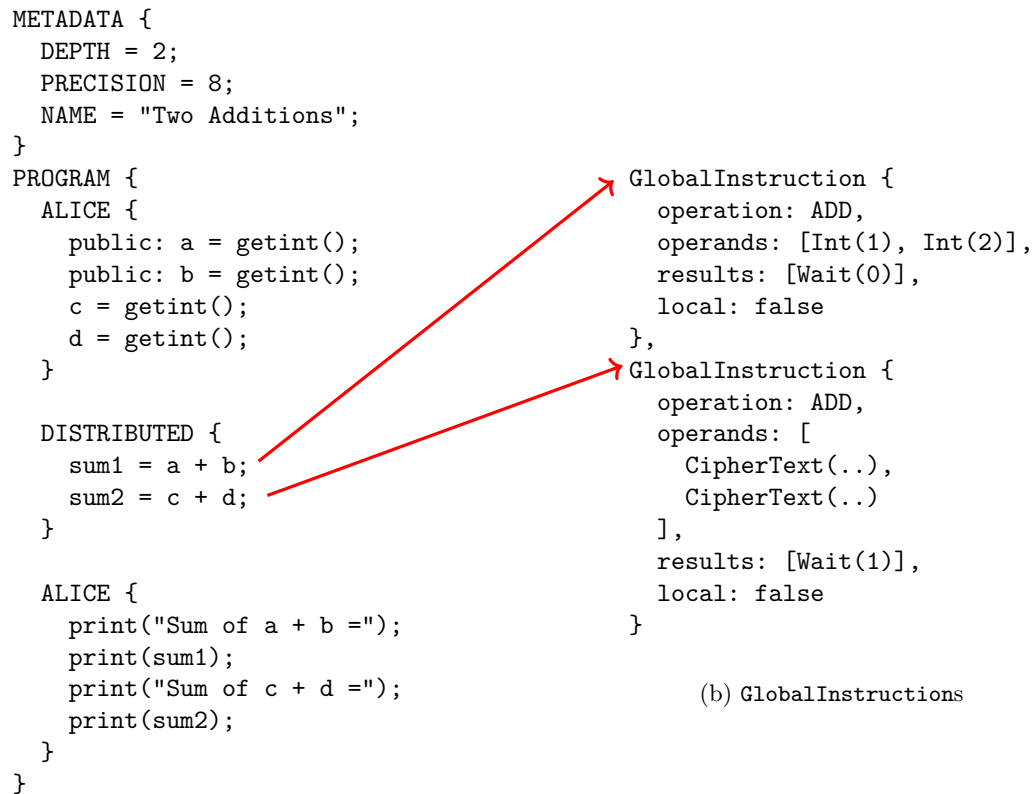
Figure 6.7: A DiSCuS program for adding two pairs of integers.

6.3.5 Implementation

The codebase is partitioned into two parts, a library with reusable elements **backend** and an executable leveraging the library called **client**. The mutation of **GlobalInstructions** and **LocalInstructions**, the interpretation of programs, and the cryptographic primitives are implemented in **backend**; whereas the distributed networking logic, the DSCS interface, and a user-facing command line interface are handled in **client**.

Later in the development process, when running end-to-end benchmarks became possible, it became clear that applications were not bottlenecked by the computational complexity of programs, but by the limitations of network bandwidth. The transmission of the *Work Fragments* took orders of magnitude longer than their evaluation. Relatively simple arithmetic instructions with trivial evaluation time took considerable time to transmit, evaluate, and then retransmit results. As *Work Fragments* are constrained by the requirement that they have a fixed finite execution time, they can contain no conditional branch instructions. Early versions of DSCS also required that all intermediate results were returned to the *Work Issuing Node*. The problem of transmitting *Work Fragments* is demonstrated by the massive performance regression over theoretical execution latency demonstrated in Section 6.4. This problem was partially solved with the addition of the **LOOP** instruction to the DSCS instruction set, as a mechanism for compressing instructions. The addition of the **LOOP** operation to the instruction set has yet to be propagated to DiSCuS. There is a clearly a trade-off in the relative computational cost and the serialised size of the target executable.

The off-the-shelf distributed networking library, **libp2p**, used for DiSCuS favoured reliability over latency. This library is not an ideal choice for constructing a low-latency computational substrate. Future works and



(a) DiSCuS DSL program (Figure 6.7)

Figure 6.8: The `GlobalInstructions` generated by processing the program defined in Figure 6.7. The instructions in (a) are related to their counterparts in (b) by red arrows.



Figure 6.9: Homomorphic preprocessing, expanding GlobalInstructions into GlobalInstructions with ciphertexts expressed as CipherTextFragments

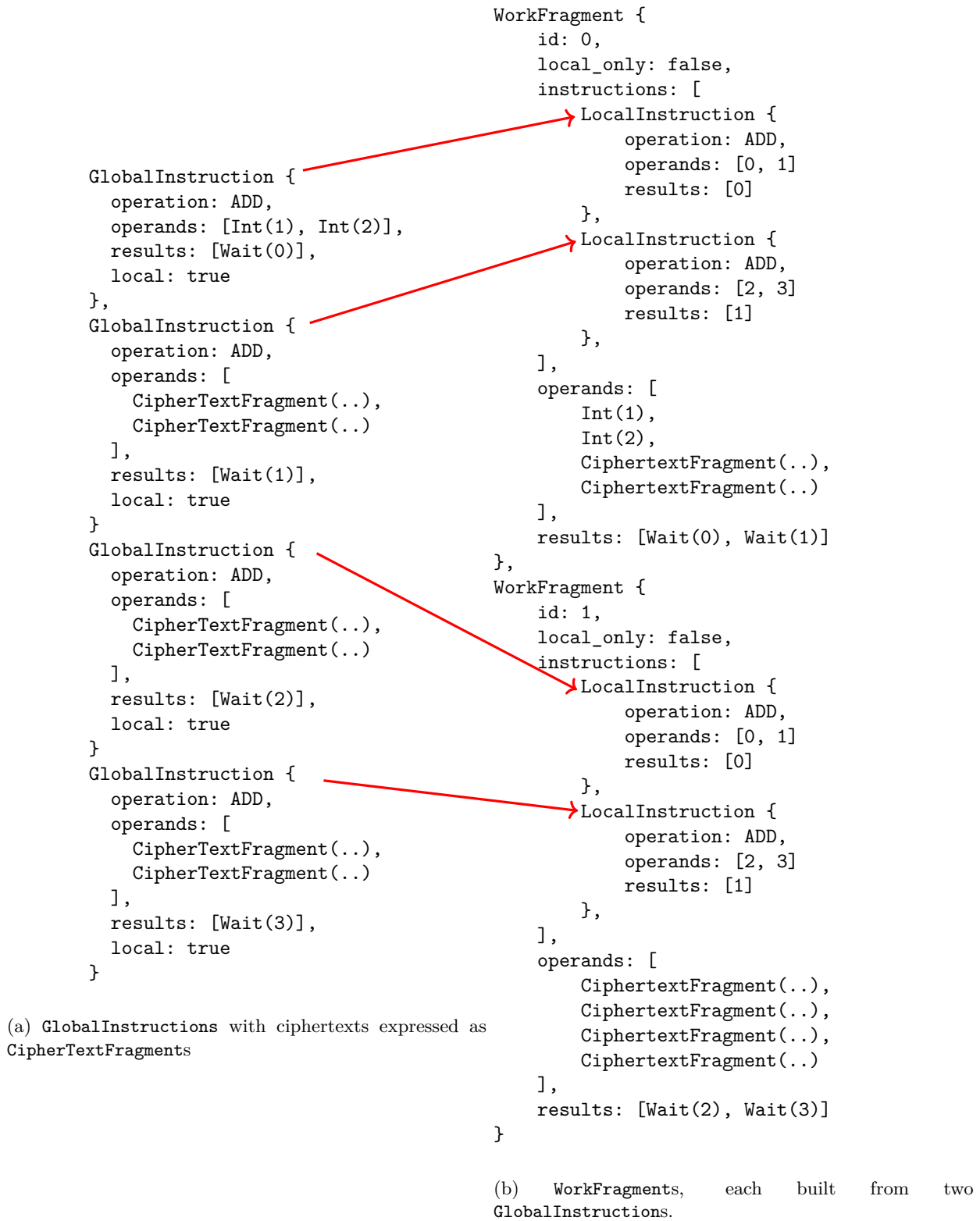


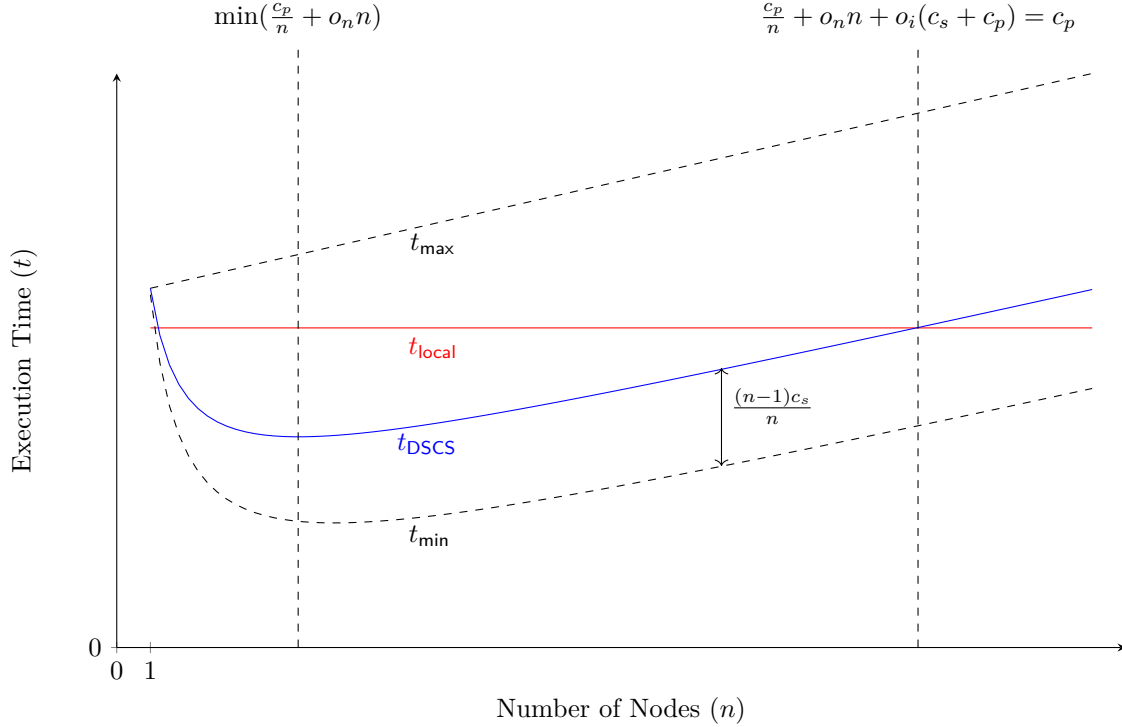
Figure 6.10: Consolidating GlobalInstructions with ciphertexts expressed as CipherTextFragments into WorkFragments

```

Result {
  id: 0,
  results: [Int(3), CiphertextFragment(...)]
},
Result {
  id: 1,
  results: [CiphertextFragment(...), CiphertextFragment(...)]
}

```

Figure 6.11: Results of outsourced WorkFragment evaluation.

Figure 6.12: Duplicate of Figure 5.33. t_{DSCS} is the target performance for DiSCuS over baseline t_{local} benchmarks.

revisions may need to develop their own networking primitives to remove this bottleneck.

6.4 Evaluation

In theory, executing a program on a DSCS network provides performance approaching the profile outlined in Figure 6.12, previously presented in Chapter 5 as Figure 5.33. Given overhead values o_n and o_i , for per-node overhead and per-instruction overhead respectively, the expected execution time of a given program is redefined as:

$$t_{DSCS} = \frac{ec_e + c_{pp}}{n} + c_{sp} + o_n n + o_i(c_p + c_s) \quad (6.6)$$

Where the conventional total program cost c is decomposed into the cost to evaluate the unencrypted sections of code designated for HE processing, c_e , the sections of code with parallelism that will be executed in plaintext form c_{pp} , and the sections of code that are fully sequential that will be executed in plaintext form c_{sp} . The

total of all the computational costs present in Equation 6.6 sum to the total cost to execute the program locally, without DSCS, $c = c_e + c_{pp} + c_{sp}$. If this equation is a reliable predictor of DSCS performance, given benchmarking results from a DSCS implementation it should be possible to extract constants that approximate program latency.

6.4.1 DiSCuS Performance

The following benchmarks were executed with a *Work Group* of multiple independent DiSCuS clients on a single desktop computer featuring an Intel i7-9700 CPU clocked at 3.0 GHz, with 8 cores. For the sake of relatively short-lived and easily repeatable benchmarks the homomorphic plaintext and ciphertext space ring dimension were set relatively low, to 16; and the remaining parameters were adaptively chosen to handle data precision requirements, and computational depth. Interesting, the complexity of ciphertext addition is linear in the ring size, so performance in these benchmarks can be expected to grow linearly for larger ring sizes. BFV addition operations are linear in the ciphertext ring dimension so reducing this factor has a linear effect on the results, which is fixed for a given benchmark.

Running the following experiments on a single computer mean the results to follow are not representative of a true DiSCuS deployment. In reality the inter-node bandwidth will be varied and network topology dependant, and the nodes themselves will have some variance in the computational resources available to them. This sequence of experiments was designed to explore DiSCuS performance under idealised conditions, and therefore control as many factors as possible. As such, the following results can be attributed to protocol design alone and not nebulous factors, such as network topology. As will quickly become evident there are some fundamental flaws with DSCS design, and as such no further experiments more representative of a true deployment are required, as DiSCuS will only perform worse as experimental conditions become less ideal.

The benchmarks results are adjusted to compensate for any Single Instruction Multiple Data (SIMD) instruction processing available, as is the case with homomorphic operations. For addition the SIMD width is the ring dimension, so whatever the ring size is set to is normalised by a proportional compensatory gain in SIMD width. The benchmark measurements were started after *Work Fragment* construction, and were stopped after the results were collected. As such they measure the time taken to send, evaluate, and collect the results of the DSCS operations and exclude any time required to preprocess operands or instructions, such as encrypting plaintext values to produce ciphertexts. A complete account of the experiment set up and raw results are provided in Appendix A.

A sequence of benchmarking operations were run, with results averaged over 30 independent experiments. Addition operations were chosen as they provide an optimistic example of DiSCuS performance without further complexity required for managing complex transformations. Furthermore, this benchmark can easily be modified to be an entirely dependent, or an entirely independent sequence of operations. Importantly, each individual homomorphic addition operation was counted as completing a sequence of operations equal to the ring order, in this case 16. This is to account for the trivial SIMD processing capabilities of HE addition.

The benchmarking programs are simple iterative permutations over a sequence of integer additions.

1. 1,000 Independent Plaintext Additions.
2. 100 Independent Homomorphic Additions, with conventionally encoded ciphertexts.
3. 100 Independent Homomorphic Additions, with DCRT encoded ciphertexts.
4. 100 Dependent Homomorphic Additions, with conventionally encoded ciphertexts.

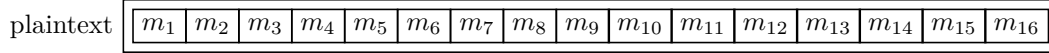


Figure 6.13: The SIMD lanes available to a BFV plaintext with ring order 16 for polynomial addition and polynomial multiplication. Each position, m_i , conceptually aligns with a polynomial coefficient of order $i - 1$.

5. 100 Dependent Homomorphic Additions, with DCRT encoded ciphertexts.

The benchmarks featuring homomorphic encryption (2-5) all leverage SIMD processing of data. This means 1600 equivalent plaintext addition operations occur per benchmark. Of note, the benchmarks featuring 100 dependent homomorphic additions (4 and 5) all actually feature 16 independent chains of 100 dependent addition operations. It is important to note here that to reflect the exploitation of SIMD HE evaluation, homomorphic additions are converted into equivalent plaintext additions by multiplying the number of executed operations by the ring dimension, and therefore this experiment actually measures the results of evaluating 16 independent sets of 100 dependent additions. The SIMD lanes available to a BFV plaintext are demonstrated in Figure 6.13, the entries in this plaintext represents polynomial coefficients of different orders. This structure is preserved throughout homomorphic evaluation, so predictable operations following the rules of polynomial arithmetic can be applied to a wide vector of data. Vectorising conventional addition is trivial, but as multiplication is executed as polynomial multiplication, multiplication involves mixing coefficients of different orders. More details on the benchmark programs and pseudocode descriptions is provided in Appendix A. The plaintext precision is set at 8 bits, operands are randomly generated, and cryptosystem parameters are generated programmatically. The bounds on parameters in RLWE-based LFHE are infamously loose, and so for a specific program it can often be useful to run a sequence of experiments to determine better program parameters, which may have a small chance of turning an error during decryption in exchange for a lighter computation cost during evaluation. This problem is tempered by the fact these benchmarks entirely consist of addition operations, which contribute to accumulating errors within the ciphertext considerably less than with multiplication. Experiments were run locally, both interpreted and compiled where the software stack allowed, and on a DSCS network with *Work Groups* of between 1 and 16 nodes.

The complete data from these experiments are articulated in the sequence of graphs that follow, and then summarised in a series of tables. Experiment 1 features 1000 plaintext addition operations, and is intended to provide a relatively optimistic picture of DSCS performance. Unfortunately, the graph in Figure 6.14, when compared with local-only execution given by c , never improves over this target. The performance improves somewhat as the initial nodes join the *Work Group*, but this reaches a stable minimum well above the local-only latency. This indicates that the computational overheads involved in supporting the DiSCuS network do not yield improvements in computational latency. The equation of predicted performance given in Equation 6.6 can be applied to this experiment. The computational cost is 0 for all computational cost variables except c_{pp} which equals the conventional computational cost. As such the expected latency is given by the following expression:

$$t_{DSCS} = \frac{c}{n} + o_n n + o_i c \quad (6.7)$$

Using polynomial regression to fit polynomial of the form given in Equation 6.7 to the experiment results yields constants $c = 720$, $o_n = 1$ and $o_i = 3$. The evaluation of t_{DSCS} with these constants is plotted on Figure 6.14 as a dotted line. This seems to predict the performance of DiSCuS in this experiment relatively well. The discrepancy between the predictive model and the low node count suggests some unaccounted for factors, and indeed this turned out to be the case, as the network interface was completely saturated at these low node counts - possibly due to the larger *Work Fragment* size. Even though the performance of DiSCuS

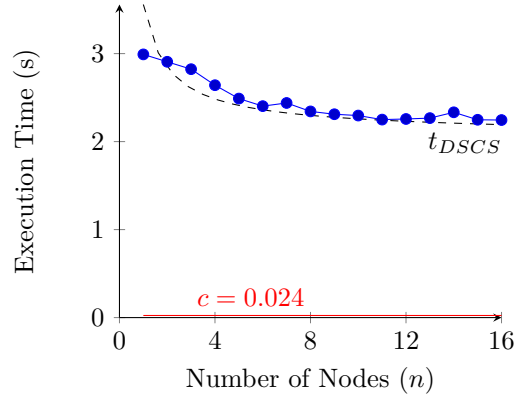
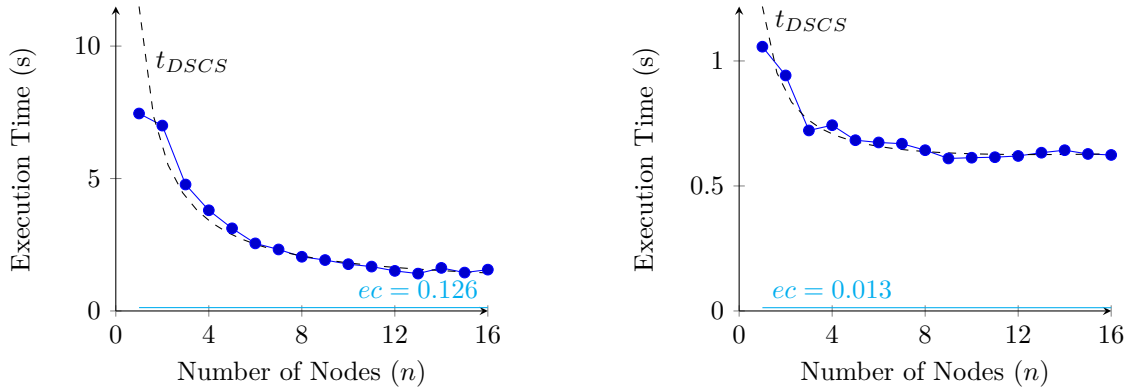


Figure 6.14: Experiment 1: Latency of 1,000 independent plaintext additions on the DiSCuS network. The line labelled c provides context of the local (interpreted) execution latency. A dotted line, t_{DSCS} , shows the predicted performance given overhead constants, defined in Table 6.4. The prediction t_{DSCS} has an R^2 value of 0.78.

in this experiment is not very compelling it is easy to see that with appropriately low values for o_n and o_i systems implementing DSCS would show promise. Ultimately the poor performance in plaintext addition is likely the result of the large ratio between the *Work Fragment* size and the computational complexity of simple addition operations.

However, this is a very specific experiment, in which a relatively small number of independent operations are performed, operations which any computer can already execute by itself with extremely low latency. Perhaps DiSCuS will have more utility in a domain classical computational substrates traditionally struggle with, namely homomorphic evaluation. Figure 6.15 shows the experimental results from executing 1600 plaintext equivalent homomorphic independent addition operations on DiSCuS. Subfigure 6.15a graphs the result of the experiment when ciphertexts are encoded conventionally as pairs of ciphertexts in with large coefficients, and Subfigure 6.15b graphs the results of the experiment when ciphertexts are DCRT encoded. For experiment 2 the predicted latency in Equation 6.6 can be simplified to the expression in Equation 6.7. Constants can be extracted as before, this time $c = 10794$, $o_n = 4$, $o_i = 0.06$. As expected the computational load is vastly different, but interestingly the per-node operational overhead is four times larger than in experiment 1, while the per-instruction operational overhead has decreased by a factor of 23 between experiments. This would indicate that the per-instruction overhead is not fixed but actually depends on the instruction in question. This makes sense as each instruction has a vastly different operand size, and as such requires more time to coordinate the distribution of *Work Fragments*. This experiment was much more receptive to acceleration via DiSCuS, supporting the claim that the efficacy of DiSCuS is tied to the ratio between *Work Fragment* serialised size and the computational complexity of individual instructions. In this way HE addition with conventionally encoded ciphertexts serves as a good counterpoint to the plaintext addition previously discussed. The obvious outlier in this experiment is at $n = 1$ which deviated from the fitted latency equation. This is most likely due to a saturated network interface. Whereas with higher node counts the *Work-Issuing Node* can distribute work to waiting *Worker Nodes* while some nodes in the *Work Group* evaluate their delegated *Work Fragments*, in the single node case the latency is directly impacted by the speed with which *Work Fragments* can be sent and results received. This problem is common across all experiments but most effectively demonstrated when the experiment takes well to DSCS parallelised execution.

For the experiment given in Subfigure 6.15a featuring operands in DCRT form, approximate constants



(a) Experiment 2: Conventional ciphertext representation. $R^2 = 0.86$. (b) Experiment 3: DCRT ciphertext representation. $R^2 = 0.92$.

Figure 6.15: Latency of 1600 equivalent-plaintext-addition-operations via independent homomorphic additions on the DiSCuS network for both conventional ciphertext representation and DCRT ciphertext representation.

can be extracted. In this case $c = 697, o_n = 4, o_i = 1.35$, which provides further evidence that the per-instruction overhead is different for each DSCS instruction. The scaling effects are less dramatic for DCRT encoded ciphertext addition, possibly because the ratio between *Work Fragment* size and the effort required to evaluate them is more dramatic with conventionally encoded ciphertexts.

The following experiments involved fully sequential homomorphic additions. As before, the two following experiments feature operands encoded as conventional BFV ciphertexts, and as DCRT encoded ciphertexts respectively. The former experiment is expected to be ill-suited for DSCS parallelism due to the undecomposable sequence of fully dependent operations, whereas the latter should demonstrate some more value in program execution on a DSCS network.

As predicted, the experiment in Subfigure 6.16a shows a regression in execution latency as more nodes are added to the *Work Group*. This is a result of the interdependency between the instructions included in the benchmark. This interdependency corresponds to a simplification in the equation for predicted latency:

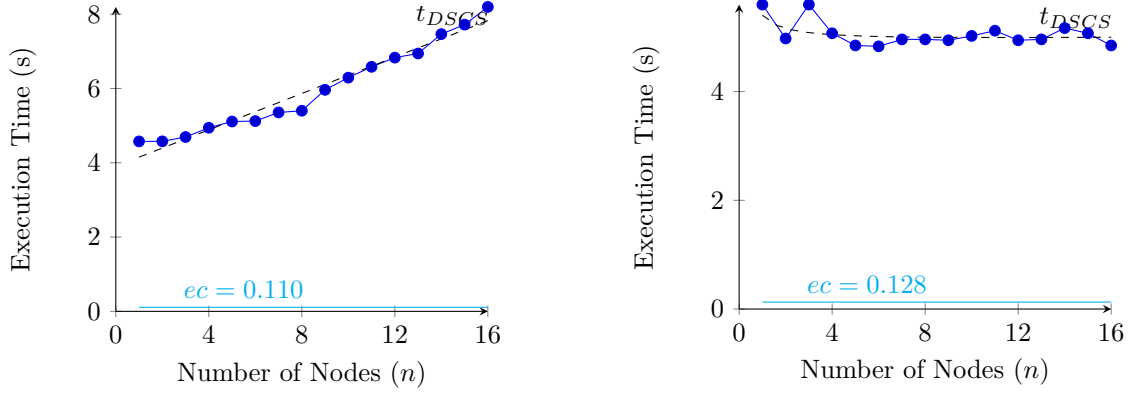
$$t_{DSCS} = c + o_n n + o_i c \quad (6.8)$$

Using polynomial regression provides estimates for the constants in this equation, however as both c and $o_i c$ have no n component, as such the coefficient of n of order 0 is $c + o_i c$. Therefore, when using polynomial regression these components are summed, and extracting individual values is impossible. In this experiment $c + o_i c = 3906$ and $o_n = 245$. As predicted, the latency increases linearly with *Work Group* size, and a *Work Group* with a single *Worker Node* has the lowest latency.

The complementary experiment featuring DCRT ciphertexts, Subfigure 6.16b, better utilises the parallelism made available by DSCS, over Subfigure 6.16a, but fails to gain much in the way of performance improvements as the *Work Group* scales in size. The predicted latency is expressed as the simplified equation:

$$t_{DSCS} = \frac{c}{n} + o_n n + o_i c \quad (6.9)$$

due to the newly introduced DCRT parallelism. The constants extracted by polynomial regression are $c = 492, o_n = 3.75$ and $o_i = 10$. This high per-instruction overhead could be caused by the effort involved in



(a) Experiment 4: Conventional ciphertext representation. $R^2 = 0.96$. (b) Experiment 5: DCRT ciphertext representation. $R^2 = 0.44$.

Figure 6.16: Latency of 1600 equivalent plaintext addition operations via dependent homomorphic additions on the DiSCuS network for both conventional ciphertext representation and DCRT ciphertext representation.

Benchmark	Equivalent Plaintext Operations per Second			
	Local (Compiled)	Local (Interpreted)	DiSCuS (1 Node)	DiSCuS (16 Nodes)
Independent Addition	67,980,000	400,123	3342	4457
Independent HE Addition	21,712	12,709	215	1,029
Independent DCRT HE Addition	-	120,544	1,516	2,578
Dependent HE Addition	21,680	14,544	353	196
Dependent DCRT HE Addition	-	12,528	4,690	5,376

Table 6.1: Benchmark figures for DiSCuS applications.

repeatedly checking and updating operand dependencies during execution.

A summary of all experimental results is given in Table 6.1, with the results normalised against Local (Interpreted) benchmarks in Table 6.2, and the isolated 16 node vs 1 node DiSCuS speed-up in Table 6.3. The metric Operations per Second refers to the number of equivalent plaintext addition operations executed per second, irrespective of whether those additions use conventional arithmetic or HE arithmetic. The performance of DiSCuS (1 Node) when compared to Local (Interpreted) reflects the pure overheads added to execution by executing a program over the DiSCuS network, as the DiSCuS (1 Node) benchmarks involve outsourcing the entire benchmark computation to a single remote node.

In conjunction with the individual benchmark details given above, Table 6.1, 6.2 and 6.3 demonstrate

Benchmark	Normalised Equivalent Plaintext Operations per Second			
	Local (Compiled)	Local (Interpreted)	DiSCuS (1 Node)	DiSCuS (16 Nodes)
Independent Addition	169.898	1	0.008	0.011
Independent HE Addition	1.708	1	0.017	0.081
Independent DCRT HE Addition	-	1	0.012	0.021
Dependent HE Addition	1.491	1	0.024	0.013
Dependent DCRT HE Addition	-	1	0.374	0.429

Table 6.2: Benchmark figures for DiSCuS applications normalised against Local (Interpreted) results.

Benchmark	Speed-up (16 nodes vs 1 node)
Independent Addition	1.33
Independent HE Addition	4.79
Independent DCRT HE Addition	1.70
Dependent HE Addition	0.56
Dependent DCRT HE Addition	1.15

Table 6.3: Speed-up of 16 node *Work Group* over 1 node *Work Group*

Benchmark	c	o_i	o_n
Independent Addition	0.72	0.003	0.001
Independent HE Addition	10.794	0.00006	0.004
Independent DCRT HE Addition	0.697	0.00074	0.004
Dependent HE Addition	3.906		0.245
Dependent DCRT HE Addition	0.492	0.01	0.00375

Table 6.4: Constants extracted from experimental results by polynomial regression.

four immediate results:

1. The interpreter has a remarkably negative effect on computational throughput.
2. When moving from a single node *Work Group* to a 16-node *Work Group* performance does not approach a 16-fold improvement. This is demonstrated more clearly by Table 6.3.
3. The computational throughput declines dramatically between local interpreted execution and single node DiSCuS networked execution. This result is demonstrated more clearly by Table 6.2.
4. DiSCuS was able to introduce a measurable speed up when distributing a fully sequential sequence of HE operations over a 16 node *Work Group*, as compared with a 1 node *Work Group*.

First, that the DSCS interpreter could undergo a great deal of improvement. As it currently stands, interpreted computations achieve an operations-per-second of between 2 times and 100 times slower than their compiled counterparts. Second, when comparing single-node DSCS *Work Group* performance to 16-node *Work Group* performance the operations-per-second did not go up 16-fold. There was a definitive and noticeable improvement in performance but this was an improvement of, at best, 5 times. This indicates the per-node operational overhead, o_n , incurred by this DSCS implementation has a non-negligible effect on the execution time. Similarly, comparing the interpreted results to the single node DiSCuS execution shows a sizeable reduction in performance. The computational power dedicated to the tasks is the same between these columns, but the computation is now networked instead of running on a conventional processing model. This indicates a larger than expected per-DSCS-instruction operational overhead o_i . Finally, these experiments show that DiSCuS was successfully able to inject parallelism into a previously fully sequential sequence of operations. In the Dependent DCRT HE Addition experiment the computational throughput increased when the previously fully sequential operations were parallelised and distributed to the 16 collaborators in the *Work Group*.

Table 6.4 contains the constants extracted by using polynomial regression to find a curve that best fits the data from each experiment, the process for which is given in more detail in Appendix B. Interestingly, the per-node overhead is relatively constant for the HE experiments. The per-instruction overhead varies

between experiments using the same type of instruction suggesting that different instructions have different per-instruction overheads.

Some of the performance problems of DiSCuS can be traced back to the underlying network primitives, which were designed for distributed communication prioritising resilience. With acceptable latency in the forgiving range of human-computer interface (such as distributed messaging applications) and not low latency edge computing. The libraries used guarantee messages will reach the target but not the time frame it will take. Unfortunately DiSCuS suffers from a minimum round trip message time of between 1–3ms per instruction. This is a dramatic delay compared to the same program running natively on a single node. Similarly, the processes farming out work and collating results are implemented with an asynchronous function library, allowing for out-of-order work completion but which halts execution until the thread handler returns to check for results.

A lot of the problems with DiSCuS appear to be low-hanging-fruit engineering work. When the issues around the ratio between *Work Fragment* size and computational complexity are resolved it is well within the realms of possibility for homomorphic evaluation, distributed across a network of mutually distrusting peers, to achieve latency which improves over local-only execution. The effect on programs that are not backed by homomorphic encryption is less clear. As long as adequate computational work can be fit into a relatively compact *Work Fragment*, and the target computation contains some degree of parallelism, distributing across a tuned DSCS implementation could well lead to large improvements in computational latency and device saturation.

6.5 Related Work

There are a number of software libraries written to support homomorphic encryption and execution. Microsoft’s SEAL [102], HEAAN [1], PALISADE [2], and more recently lattigo [3], each offer a variety of homomorphic cryptosystems with varying functionality. Work accelerating homomorphic encryption tends to use specialised hardware designs deployed on FPGAs [13, 40, 45, 46, 75, 105]. These designs largely leverage DCRT representations of ciphertexts to parallelise HE operations. DiSCuS has drawn from these works for inspiration. The FPGA works featuring DCRT representations in particular share a lot of implementation choices with DiSCuS. Where they use the parallelism injected by DCRT to better utilise FPGA resource, in DiSCuS this parallelism is used to drive multiple disparate collaborating nodes.

There are existing distributed computation frameworks, these require a trusted authority to provide computational tasks and can offer no explicit incentive for participation, as such they target volunteer scientific computation for the public good [9, 18]. Even while only implementing the least secure DSCS specification, DiSCuS operates on a fundamentally different security assumption than these existing works. Namely, DiSCuS provides a high degree of data confidentiality. More in-depth discussion around these existing distributed computation frameworks featured earlier in this thesis, in Subsection 2.2.1, and then in relation to the DSCS in Section 5.2. To the best of the authors knowledge there are no existing distributed computation frameworks using homomorphic encryption to allow mutually distrusting nodes to operate on private data.

Blockchain-backed computation offers some of the notions presented in this work. Unlike this work it provides guarantees about programs executing regularly and reliably, but is uninterested in computational throughput nor efficient resource utilisation. The distributed nodes of blockchain-backed computation are used to provide resilience with distributed consensus over the state. These design decisions are largely orthogonal to DiSCuS, which is trying to accelerate target computations through cooperation between nodes

with some degree of mutual distrust.

6.6 Further Work

6.6.1 Runtime

There are substantial and easy gains in computational latency to be made in DiSCuS’s DSCS interpreter. With relatively straight forward engineering work optimisations can be made to the way DSCS instructions are executed leading to substantial reductions in the per-instruction overhead, o_i . Throughout the execution of a *Work Fragment* the interpreter repeatedly reads to, and writes to a single data structure, exhaustively checking dependencies, and insuring *Work Fragment* validity. These checks can be mostly omitted. Similarly, rebuilding the DiSCuS network stack to use high performance bespoke distributed network primitives would certainly similarly reduce the per-instruction overhead by a demonstrable degree.

6.6.2 Compiler

Some run-time processing could be offloaded to a compilation step. A host of execution-level optimisations would massively improve computational throughput. The propagation of operation results to dependent operations is also a needlessly complex step that could be improved by having a format in which operations indicate which operations they unblock, rather than requiring an exhaustive search across all operands to find which operands are awaiting a newly produced result. Currently optimisations must be done by hand and making use of the SIMD nature of HE evaluation requires manual intervention on the part of the programmer, this step could easily be offloaded to a compiler. Beyond these changes, integration with a cryptographic compiler such as Viaduct [4] would allow for a more pleasant developer experience, provide preprocessing steps to better manage data dependencies, and would also allow for automatic optimisations during the generation of the DSCS-compatible programs.

6.6.3 DSCS Protocol

DiSCuS only supports an outdated version of DSCS in the lowest security model. More work could see the remaining functionality described in the previous chapter actualised and evaluated qualitatively. Currently BFV multiplication also resists parallelisation due to the sequential components before and after the double CRT multiplication. Tighter integration with this process could yield performance benefits. The DSCS *Work Fragment* format could be updated to support a smaller data pool by allowing data to be repeatedly written to a single location. This would require reworking the assumptions around dependency management, but this work would pair well with any further work rewriting result propagation logic and would reduce the per-instruction overhead of programs evaluated on DiSCuS. Similar gains could be achieved by reimagining the DSCS execution model as an abstract stack machine or another more concise program encoding. Unfortunately, any deviation from arithmetic circuits causes the protocol to forgo the ease of extensibility with the wide variety of advanced cryptographic primitives using arithmetic circuits to model general computation.

6.6.4 Speculative Research

DSCS programs are not themselves decentralised, as currently an ephemeral centralised sub-network is constructed from the distributed DSCS network to execute a specific task. DSCS primitives could be developed

to support network-level ownership of executables, where programs are organically proposed and corroborated by nodes on a distributed network to create true decentralised applications. For example, perhaps a swarm of nodes run regular health checks on their peers which are spawned semi-randomly but only if they haven't participated in a health check for a given amount of time.

6.7 Conclusion

DiSCuS implements an early version of the DSCS protocol as specified in the lowest security model. It allows a disparate network of edge devices to collaborate and operate on secure data in a way not possible before. By leveraging a group of devices, computational load can be shared between devices, opening the possibility of homomorphic evaluation on edge devices. This dramatically reduces the attack surface, both physical and digital, a fact indispensable in many domains.

Unfortunately there is a large amount of work to be done in improving DiSCuS, and modifying the DSCS definition, to produce a compelling computational substrate for edge devices to securely collaborate on tasks. Under these conditions there is no reason for a device to invest the effort into organising a *Work Group* of peers over performing a computation locally, in isolation.

However, there are a slew of low-hanging-fruit improvements that can be made to both DSCS and DiSCuS, which in conjunction may well result in massively reduced per-node and per-instruction overheads. It is certainly within the realm of plausibility that a system like DiSCuS could provide performance good enough to encourage use over local-only execution. Furthermore, DiSCuS clearly demonstrates that a system for decentralised privacy-preserving computations in an environment of untrusted nodes can be built. It is hard to estimate the maximum throughput expected by a distributed computation protocol implementation of the nature of DiSCuS, but it is not hard to expect latency to fall well below local-only evaluation of programs backed by homomorphic encryption. This goal becomes especially attainable if *Work Fragments* are made compact enough that their expressiveness dwarfs their size. This expressiveness can be achieved by expanding the DSCS instruction set. Coupling DSCS improvements with further engineering work to optimise the DiSCuS runtime and compiler, it is more than conceivable that expected latency could fall well below the time taken by local-only evaluation.

Chapter 7

Conclusion

This thesis began with the overarching goal of building machinery to allow for secure decentralised and distributed computation. This required achieving a sequence of relatively independent objectives:

1. Ensure the correctness of outsourced computations, with better-than-state-of-the-art guarantees for computational complexity in the case of one-shot computations.
2. Provide apparatus for two mutually distrusting parties to fairly exchange digital information without a trusted third party intermediary.
3. Define the set of behaviours and security considerations for a platform supporting decentralised computation. Then realise these properties in a candidate protocol.
4. Demonstrate the efficacy of an decentralised computation platform, bound the engineering targets required for a compelling implementation, and guide further development work in this area.

7.1 Objectives

The overarching objectives of this thesis have a 1-to-1 correspondence to the contributions of each of the core content chapters in this thesis.

1. Proofless Verifiable Computation from Integer Factoring.
2. Two Party Fair Exchange.
3. DSCS: A Trusted Computation Mesh.
4. DiSCuS: A Partial DSCS Implementation.

The following sections tackle each of the aforementioned content chapters one-by-one, reiterating the objectives, contributions, and the scope for improvement in further work.

7.1.1 Proofless Verifiable Computation from Integer Factoring

The class of VC protocols allow for a Verifier to validate the correctness of the results of a computational problem outsourced to a Prover. Prior to this work all number theoretic VC constructions required some

accompanying proof of correctness with any results, and usually a computationally expressive program pre-processing step, which often carried a greater computational cost as the target computation. To the best of the authors knowledge no existing works relied on the well studied complexity assumption relating to the intractability of integer factoring and often instead relied on less well studied complexity assumptions. A sequence of new VC protocols were presented in Chapter 3, culminating in a single protocol which offers a compelling alternative to state of the art VC due to the improved worst-case computational complexity and a novel additional property: prooflessness. Despite the novelty of this scheme, the protocol is founded on a standard and well-studied cryptographic assumption - that of the hardness of integer factoring. This work provides a compelling VC scheme with a guaranteed worst-case performance, that is no worse than duplicating the execution of a program locally, a feature not shared by related work.

The final protocol presented in Chapter 3:

- Is *Proofless*: the results provided by a Prover include no information beyond the raw output of a target computation as if it was being outsourced without verifiable computation.
- In Prover-optimal, the Prover does no more work than if they were evaluating an unverified outsourced computation.
- Is built from a standard and well-studied cryptographic assumption: the hardness of integer factoring.
- Satisfies standard VC security properties of *Completeness*, *Soundness*, and is always *Outsourceable*.
- Is composed of small one-time setup and a zero cost program pre-processing step..
- For computations with large intermediate operands has a verification step with computational complexity that is a constant factor of the computational complexity of the target program.

The protocol features lead to trade offs compared to related work:

- The fraction of work done by the Prover is based on the relationship between the size of the integer factors in the foundational integer factoring problem and the size of the intermediate operands of the target computation. As such, the cost to the Verifier increases as the security parameter λ increases. Verification may cost the same, but no more, than evaluating the computation locally.
- Related work often features a constant cost verification step, whereas in this protocol the verification scales with the complexity of the target computation.

These compromises show that this protocol exists in its own niche relative to the related work. It is *Proofless*, satisfies strong security guarantees, and is always *Outsourceable* for computations with large intermediate operands. It is particularly well suited to verifying the one-shot evaluation of large integer arithmetic, a primitive used extensively in lattice cryptography, and by extension many contemporary HE schemes.

7.1.2 Two Party Fair Exchange

The class of FE protocols allow two mutually distrusting participants to exchange secret information where each parties ability to receive a secret is contingent on sending their own secret. The previous state of the art required the *optimistic* use of a trusted third party, such that the third party need only get involved in the case of a dispute [79] . The work presented in Chapter 4 culminated in the definition of a FE scheme

which, to the best of the authors knowledge, is the only FE scheme to simultaneously satisfy a notion of strong fairness without requiring any intervention from a trusted third party. This work utilised a number of advanced modern cryptographic primitives, and took common approaches from the field of consensus to be applied to FE, which is known to be a related problem. This work provides a counter example to overturn a commonly cited belief that FE satisfying a notion of strong fairness must involve a third party in some capacity.

The final protocol presented in Chapter 4:

- Satisfies the strong security notions of *Computational Strong Fairness* and *Computational Completeness*.
- Is the first FE protocol satisfying a notion of strong fairness to not require a trusted third party intermediary.
- Has a preconfigured constant probability of round-by-round success.

Unfortunately, the ability to avoid a trusted third party has introduced a number of less desirable qualities:

- The preconfigured constant probability of success must usually be configured to be very low to reduce the cost of recovering from cheating. This means the protocol is expected to terminate after a constant, but very large number, of repeat rounds.
- Uses expensive cryptographic primitives to support bootstrapping security from semi-honest adversaries to arbitrarily malicious adversaries.
- Is restricted to the domain of “secure enough”.

Despite the scope for improvement this protocol is a first-of-its-kind and overturns a widely held belief in the field of cryptography. Despite acceptable asymptotic qualities, the large constant number of protocol rounds required to successfully exchange secrets severely limits the practical use of the two party FE presented in this work. Despite these less than favourable qualities it is the *only* scheme that allows the secure exchange of secrets between two mutually distrusting parties without the involvement of a trusted third party arbitrator.

7.1.3 DSCS: A Trusted Computation Mesh

The work in Chapter 5 defines a new class of protocol, a TCM, which allows a decentralised distributed network of devices to organically cooperate on the execution of decentralised computational tasks. The TCM definition was accompanied by an example protocol, DSCS, alongside extensions to reinforce the security arguments in the face of a series of progressively more powerful adversaries. The DSCS protocol extensions made extensive use of modern cryptographic primitives, including those presented in Chapter 3 and Chapter 4.

This work successfully:

- Formalised the class of SMUS’, a class of computational substrates supporting notions of computational security.
- Restricted the definition of an SMUS in order to define new class of protocol, the TCM, supporting a computational substrate distributed over many devices.
- Provided security definitions and proof frameworks for the new classes of protocol, SMUS’ and TCMs.
- Defined a novel secure tokenised credit scheme with a small data footprint.

- Proposed a candidate TCM, DSCS, which with extensions formally satisfies the definitions of a TCM in the face of the classes of adversary:
 - *Nosy Nodes*, unable to deviate from the protocol but interested in viewing private data. This is the default DSCS behaviour.
 - *Lazy Nodes*, who will only engage with the network to maximise their utility. This was achieved with the addition of the credit scheme to drive tit-for-tat evaluation of work.
 - *Untrusted Nodes*, who will actively deviate from the protocol to gain some advantage. This required the composition of DSCS in the *Lazy Nodes* model with VC protocols and two party FE.
- The performance metrics required for DSCS to be a favourable alternative to conventional local-only computation were defined.

Some areas for further research were identified as part of the work completed in this chapter:

- Currently VC and FE resist composition due to fundamental differences in their problem definitions. Producing a scheme which satisfies both sets of properties is not a trivial task.
- DSCS supports no notion of distributed file storage, and as such data is firmly centralised at the originating node. This would allow an equivalent class of programs to the persistent smart contract hosted on a blockchain.
- Some properties are backed by non-rigorous heuristic arguments instead of formal proofs.

7.1.4 DiSCuS: A Partial DSCS Implementation

DiSCuS, as presented in Chapter 6 is a partial DSCS implementation used to validate theoretical performance claims made about DSCS and suggest protocol improvements that can be fed back into the DSCS specification. The implementation focused on an early revision of the DSCS protocol specification in the *Nosy Nodes* security model, and as such featured a smaller subset of supported DSCS functionality. As part of this work the BFV LFHE cryptosystem was implemented and deeply integrated with the DSCS instructions, as per the DSCS specification.

The work completed as part of Chapter 6 includes:

- Providing details for a DSCS reference implementation, including the engineering challenges and factors for implementer consideration.
- Including a comparison of candidate LFHE cryptosystems, with a recommended choice for DSCS implementation.
- Detailing the close integration of LFHE with DSCS, including how this interacts with the DCRT representation of ciphertexts in BFV.
- Expanding the process by which high-level programs are compiled into DSCS compatible *Work Fragments*.
- Performing experiments and evaluating the performance measurements of DiSCuS accelerated program execution.

This work was successful in highlight fundamental flaws in the DSCS specification, and there remains work to be done in improving the execution latency of target programs. Benchmarks provided in Chapter 6 routinely failed to improve over local-only execution. The performance, in general, conformed to predicted performance estimations but with overheads in excess of tolerable limits. Ultimately, the issues observed can be largely attributed to engineering problems, solutions to which include:

- Further compiler improvements:
 - The automatic optimisation of programs, which are currently directly translated into DSCS instructions.
 - Automatic leveraging of BFV SIMD lanes.
 - Integration with a compiler such as Viaduct [4] to allow for more high level abstraction for a programmer.
- Further runtime improvements:
 - Omitting *Work Fragment* validity checks, in favour of bounding execution with agreed costs in order to reduce best-case throughput.
 - Improving the dependency management of operations.
- Further DSCS protocol improvements:
 - DSCS *Work Fragment* instruction set extensions to provide more expressive and compact *Work Fragments*, possibly deviating from arithmetic circuits.
 - Allowing multiple-write data pools in *Work Fragments* to compress *Work Fragment* size, or possibly overhauling the DSCS instruction set and interpreter to operate as an abstract stack machine.

Ultimately the viability of DSCS relates to the ratio between the serialised size of a *Work Fragment* and the computational complexity of the program it is encoding. In order to construct *Work Fragments* with a compactness that better aligns with a DSCS computational substrate, DSCS must support more complex instructions in the DSCS *Work Fragment* instruction set. This would have ramifications for the negotiation of credit between parties, as the *Work Issuing Node* might not be privy to the outcome of more complex intermediate instructions, such as conditional branching. Possibly these problems could be mitigated by promoting the use of VC from the *Untrusted Node* security model to the *Lazy Node* security model, but this is speculation and would require further research. Choosing a program representation that deviates from arithmetic circuits would have wide reaching consequences relating to protocol extensibility and compatibility with high level cryptographic constructions.

7.2 Towards Secure Distributed Computation

As each chapter directly relates to one of the stated objectives of this thesis, the successful achievement of the contributions in each content chapter are reflected in the overall successful attainment of the goals of this thesis. In Chapter 3 a new VC protocol was suggested, with some novel properties, build from well studied primitives which has performance metrics offering a complementary alternative to the state-of-the-art in the field. In Chapter 4 a first-of-its-kind FE scheme was defined, which required no trusted third party involvement, and the existence of which overturns a widely held belief in the field. In Chapter 5 the

problem of secure decentralised distributed computation was formalised, and a candidate protocol, DSCS, was presented with modification to suit a variety of threat models, the strongest of which made use of the previous VC and FE contributions. In Chapter 6 a partial DSCS implementation was outlined, with implementation complexities detailed, as well as benchmarks and clear signposts to a deep well of fruitful further research. These chapters in conjunction relate to a successful unified attempt to construct protocols and technologies that address some of the many problems facing secure decentralised distributed computation. Purely theoretic cryptographic constructions with formal proofs of security act as a skeleton, upon which engineering work and implementation has provided insight into the challenges faced in solving the stated problem. Furthermore, this work highlights some engineering lessons learned in pursuit of these goals, with clear guidance on how these can be avoided in subsequent research activity.

7.3 Challenges

This work was not without a fair share of challenges. At the offset, it was not clear how gaps in cryptographic literature would be filled. For example, two party FE was widely believed to be impossible and overturning this belief would be no simple task. Similarly, there was no clear process for reducing the set up costs of VC schemes, as all contemporary VC schemes have an expensive set up phase which must be amortised over repeat executions. In fact, the VC scheme presented in this thesis diverged heavily from the related work, and was built from a well-studied primitive completely novel to the VC setting.

The work in this thesis also defines a new class of cryptographic construction, the SMUS and a subclass of this, the TCM. The definitions of which needed intuitive descriptions that unambiguously satisfy conventional ideas of security in the setting, while also easing the development of rigorous proofs for given construction. So, a great deal of work had to be completed before the first constructions were conceived to ensure the problem was well defined. Foundational work of this nature has an inherent tension between maximising security properties and attainable formal constructions.

The scope of the engineering work required by the DSCS implementation, DiSCuS, is not to be understated. Implementing LFHE is inherently problematic as any attempts at debugging opaque ciphertext operands are non-trivial, and the large software framework took considerable man-hours to fabricate. As such it does not implement the full suite of DSCS functionality, as to do so would add an unworkable time burden to this PhD.

7.4 Future Work

Of the tasks highlighted individually for each chapter above, the most important and impactful for further improvements in the domain of secure distributed computation are:

- The composition of VC and FE.
- The construction of an optimised TCM, building on from DSCS.

The former is essential to allowing the construction of non-theoretical TCMs in untrusted environments. The latter would involve realising the slew of DSCS optimisations which constitute relatively low hanging fruit, and then reinventing the protocol with extensions to support distributed file storage, and more expressive *Work Fragments*. Ultimately, when the *Work Fragments* are expressive enough to have a computational

complexity that far outstrips their size the network latency, the issues discovered in DiSCuS will be rendered largely inconsequential.

7.5 Conclusion

This body of work includes a novel VC construction, an FE construction previously thought to be impossible, a new class of cryptographic construction supporting secure decentralised distributed computation, and a candidate protocol which was designed, proven secure, and evaluated against an implementation. Each of these contributions not only stand on their own, but also support each other and highlight strengths in these technologies. Each contribution intends to solve a problem in the domain of secure decentralised distributed computation, and in conjunction with the lessons learned and suggestions for further work, the stated goal of secure decentralised distributed computation is brought into sharper focus for the benefit of future researchers. This work marks a step towards constructing secure protocols supporting decentralised distributed collaborative computation in a swarm of mutually distrusting peers.

Bibliography

- [1] HEAAN Homomorphic Encryption for Arithmetic of Approximate Numbers. <https://github.com/snucrypto/HEAAN>, May 2020.
- [2] PALISADE Lattice Cryptography Library (release 1.9.2). <https://palisade-crypto.org/>, April 2020.
- [3] Lattigo: Lattice based multiparty homomorphic encryption library in Go. <https://github.com/tuneinsight/lattigo>, Aug. 2022.
- [4] C. Acay, R. Recto, J. Gancher, A. C. Myers, and E. Shi. Viaduct: An extensible, optimizing compiler for secure distributed programs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 740–755, New York, NY, USA, 2021. Association for Computing Machinery.
- [5] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, C. Miller, D. Moody, R. Peralta, et al. Status report on the third round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2022.
- [6] M. Albrecht, S. Bai, and L. Ducas. A subfield lattice attack on overstretched ntru assumptions. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 153–178, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
- [7] S. Alsouri, Ö. Dagdelen, and S. Katzenbeisser. Group-based attestation: Enhancing privacy and management in remote attestation. In A. Acquisti, S. W. Smith, and A.-R. Sadeghi, editors, *Trust and Trustworthy Computing*, pages 63–77, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [8] M. Amico, Z. H. Saleem, and M. Kumph. Experimental study of Shor’s factoring algorithm using the IBM Q Experience. *Phys. Rev. A*, 100:012305, Jul 2019.
- [9] D. Anderson. BOINC: A system for public-resource computing and storage. In *Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10, 2004.
- [10] B. Applebaum, N. Haramaty, Y. Ishai, E. Kushilevitz, and V. Vaikuntanathan. Low-complexity cryptographic hash functions. Cryptology ePrint Archive, Paper 2017/036, 2017. <https://eprint.iacr.org/2017/036>.
- [11] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In S. Abramsky, C. Gavaille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, editors, *Automata, Languages and Programming*, pages 152–163, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

- [12] M. J. Atallah and K. B. Frikken. Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS '10*, page 48–59, New York, NY, USA, 2010. Association for Computing Machinery.
- [13] A. Aysu, C. Patterson, and P. Schaumont. Low-cost and area-efficient FPGA implementations of lattice-based cryptography. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 81–86. IEEE, 2013.
- [14] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, STOC '91*, page 21–32, New York, NY, USA, 1991. Association for Computing Machinery.
- [15] J. E. Baldeschwieler, R. D. Blumofe, and E. A. Brewer. ATLAS: An infrastructure for global computing. In *EW 7*, 1996.
- [16] A. Baratloo, M. Karaul, Z. Kedem, and P. Wijckoff. Charlotte: Metacomputing on the web. *Future Generation Computer Systems*, 15(5):559–570, 1999.
- [17] M. Barborak, A. Dahbura, and M. Malek. The consensus problem in fault-tolerant computing. *ACM Comput. Surv.*, 25(2):171–220, jun 1993.
- [18] 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*, pages 1–8, 2009.
- [19] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya. Incentivizing outsourced computation. In *Proceedings of the 3rd International Workshop on Economics of Networked Systems*, NetEcon '08, page 85–90, New York, NY, USA, 2008. Association for Computing Machinery.
- [20] M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 236–250, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
- [21] M. Bellare and P. Rogaway. *Introduction to Modern Cryptography*. 2005.
- [22] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, page 27–30, New York, NY, USA, 1983. Association for Computing Machinery.
- [23] M. Blum. How to exchange (secret) keys. *ACM Trans. Comput. Syst.*, 1(2):175–193, may 1983.
- [24] M. Blum and S. Kannan. Designing programs that check their work. *J. ACM*, 42:269–291, 01 1995.
- [25] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.
- [26] D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 56–73, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

- [27] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 149–168, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [28] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In J. Kilian, editor, *Theory of Cryptography*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [29] G. Brassard, C. Crepeau, and J.-M. Robert. All-or-nothing disclosure of secrets. In A. M. Odlyzko, editor, *Advances in Cryptology — CRYPTO’ 86*, pages 234–238, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.
- [30] T. Brecht, H. S. Sandhu, M. Shan, and J. Talbot. ParaWeb: towards world-wide supercomputing. In *EW 7*, 1996.
- [31] J. P. Buhler, H. W. Lenstra, and C. Pomerance. Factoring integers with the number field sieve. In A. K. Lenstra and H. W. Lenstra, editors, *The development of the number field sieve*, pages 50–94, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.
- [32] V. Buterin. A next generation smart contract & decentralized application platform, 2013.
- [33] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang. Combining ghost and casper, 2020.
- [34] 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>.
- [35] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS ’17*, page 229–243, New York, NY, USA, 2017. Association for Computing Machinery.
- [36] S. Chatterjee, N. Koblitiz, A. Menezes, and P. Sarkar. Another look at tightness II: Practical issues in cryptography. In R. C.-W. Phan and M. Yung, editors, *Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology*, pages 21–55, Cham, 2017. Springer International Publishing.
- [37] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.
- [38] S. Cheshire and M. Krochmal. Multicast DNS. <https://www.rfc-editor.org/info/rfc6762>, 2013.
- [39] K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In T. Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 483–501, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [40] A. Cilardo and D. Argenziano. Securing the cloud with reconfigurable computing: An FPGA accelerator for homomorphic encryption. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1622–1627. EDAA, 2016.

- [41] 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*, 57(6):365–388, 2020.
- [42] B. Cohen. The BitTorrent protocol specification. https://www.bittorrent.org/beps/bep_0003.html, 2008.
- [43] Cormen, Leiserson, Rivest, and Stein. *Introduction to Algorithms (Second ed.)*. MIT Press and McGraw-Hill, 2001.
- [44] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS '12*, page 90–112, New York, NY, USA, 2012. Association for Computing Machinery.
- [45] D. B. Cousins, K. Rohloff, C. Peikert, and R. Schantz. An update on SIPHER (Scalable Implementation of Primitives for Homomorphic EncRyption) - FPGA implementation using simulink. In *2012 IEEE Conference on High Performance Extreme Computing*, pages 1–5. IEEE, 2012.
- [46] D. B. Cousins, K. Rohloff, and D. Sumorok. Designing an FPGA-accelerated homomorphic encryption co-processor. *IEEE Transactions on Emerging Topics in Computing*, 5(2):193–206, 2017.
- [47] C. Dahdah, C. V. Leeuwen, Z. Kheil, J. Lacan, J. Detchart, and T. Gateau. Enabling monetization of depreciating data on blockchains. In *International Conference on Information Systems Security and Privacy*, 2021.
- [48] A. Dalton, D. Thomas, and P. Cheung. A novel preprocessing-free proofless verifiable computation scheme from integer factoring. Cryptology ePrint Archive, Paper 2023/586, 2023. <https://eprint.iacr.org/2023/586>.
- [49] A. Dalton, D. Thomas, and P. Cheung. Secret swapping: Two party fair exchange. Cryptology ePrint Archive, Paper 2023/585, 2023. <https://eprint.iacr.org/2023/585>.
- [50] J. R. Douceur. The Sybil Attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.
- [51] L. Eckey, S. Faust, and B. Schlosser. OptiSwap: Fast optimistic fair exchange. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, ASIA CCS '20*, page 543–557, New York, NY, USA, 2020. Association for Computing Machinery.
- [52] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.
- [53] E. Fernandes, J. Jung, and A. Prakash. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 636–654. IEEE, 2016.
- [54] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.
- [55] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, apr 1985.

- [56] S. M. Gainsbury and A. Blaszczynski. How blockchain and cryptocurrency technology could revolutionize online gambling. *Gaming Law Review*, 21(7):482–492, 2017.
- [57] R. Gennaro. Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 220–236, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [58] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 465–482, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [59] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.
- [60] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan. Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin. In A. Kiayias, editor, *Financial Cryptography and Data Security*, pages 321–339, Cham, 2017. Springer International Publishing.
- [61] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan. Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin. In A. Kiayias, editor, *Financial Cryptography and Data Security*, pages 321–339, Cham, 2017. Springer International Publishing.
- [62] D. Goldschlag, M. Reed, and P. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):39–, 1999.
- [63] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 2008.
- [64] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC ’85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.
- [65] P. Golle and I. Mironov. Uncheatable distributed computations. In D. Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, pages 425–440, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- [66] Y. Gong, Y. Jin, Y. Li, Z. Liu, and Z. Zhu. Analysis and comparison of the main zero-knowledge proof scheme. In *2022 International Conference on Big Data, Information and Computer Network (BDICN)*, pages 366–372, 2022.
- [67] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 321–340, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [68] L. Guo, Y. Fang, M. Li, and P. Li. Verifiable privacy-preserving monitoring for cloud-assisted mHealth systems. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1026–1034, 2015.
- [69] C. G. Günther. An identity-based key-exchange protocol. In *Advances in Cryptology - EUROCRYPT ’89, Workshop on the Theory and Application of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings*, volume 434 of *Lecture Notes in Computer Science*, pages 29–37. Springer, 1989.

- [70] S. Halevi, Y. Polyakov, and V. Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In M. Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 83–105, Cham, 2019. Springer International Publishing.
- [71] J. Y. Halpern and M. O. Rabin. A logic to reason about likelihood. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 310–319, New York, NY, USA, 1983. Association for Computing Machinery.
- [72] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.
- [73] M. Jakobsson. Ripping coins for a fair exchange. In L. C. Guillou and J.-J. Quisquater, editors, *Advances in Cryptology — EUROCRYPT '95*, pages 220–230, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- [74] S. Janin, K. Qin, A. Mamageishvili, and A. Gervais. Filebounty: Fair data exchange. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 357–366, 2020.
- [75] M. Khairallah and M. Ghoneima. Tile-based modular architecture for accelerating homomorphic function evaluation on fpga. In *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, volume 0, pages 1–4. IEEE, 2016.
- [76] K. M. Khan and Q. Malluhi. Establishing trust in cloud computing. *IT Professional*, 12(5):20–27, 2010.
- [77] S. King and S. Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19(1), 2012.
- [78] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric. All things considered: An analysis of {IoT} devices on home networks. In *28th USENIX security symposium (USENIX Security 19)*, pages 1169–1185, 2019.
- [79] A. Küpçü and A. Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, 56(1):50–63, 2012.
- [80] A. K. Lenstra. *Integer Factoring*, pages 611–618. Springer US, Boston, MA, 2011.
- [81] 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>.
- [82] M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 11–22, 1983.
- [83] D. Luo, Z.-G. Qin, Y. Zhao, R. Gao, and J.-H. Wang. An optimistic fair exchange protocol for P2P e-procurement. *The 2010 International Conference on Apperceiving Computing and Intelligence Analysis Proceeding*, pages 155–158, 2010.

- [84] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [85] V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for Ring-LWE cryptography. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 35–54, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [86] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*, chapter Chapter 8.4 ElGamal public-key encryption. CRC Press, 2018.
- [87] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, page 31–42, New York, NY, USA, 2016. Association for Computing Machinery.
- [88] F. Monrose, P. Wyckoff, and A. D. Rubin. Distributed execution with remote audit. In *NDSS*, 1999.
- [89] S. Nakamoto. BitCoin: A peer-to-peer electronic cash system, 2008.
- [90] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4:151–158, 1989.
- [91] E. B. (NIST), L. C. (NIST), A. R. (NIST), A. V. (NIST), R. D. (NSA), and S. S. (NSA). Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography. Standard, National Institute of Standards and Technology, 2014.
- [92] K. J. O'Dwyer and D. Malone. Bitcoin mining and its energy footprint. In *25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CICT 2014)*, pages 280–285, 2014.
- [93] H. Pagnia and F. C. G. Darmstadt. On the impossibility of fair exchange without a trusted third party. 1999.
- [94] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In A. Sahai, editor, *Theory of Cryptography*, pages 222–242, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [95] 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.
- [96] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, apr 1980.
- [97] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, apr 1980.
- [98] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, STOC '05*, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.
- [99] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, Feb. 1978.

- [100] N. Ron-Zewi and R. D. Rothblum. Proving as fast as computing: Succinct arguments with constant prover overhead. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, page 1353–1363, New York, NY, USA, 2022. Association for Computing Machinery.
- [101] A. W. Roscoe and P. Y. A. Ryan. Auditable pakes: Approaching fair exchange without a TTP. In *Security Protocols Workshop*, 2017.
- [102] Microsoft SEAL (release 3.5). <https://github.com/Microsoft/SEAL>, apr 2020. Microsoft Research, Redmond, WA.
- [103] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *Network & Distributed System Security Symposium (NDSS)*, February 2012.
- [104] P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.
- [105] S. Sinha Roy, K. Jarvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede. HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation. *IEEE Transactions on Computers*, 67(11):1637–1650, 2018.
- [106] N. P. Smart. *Cryptography made simple*. Springer, 2016.
- [107] 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.
- [108] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li. Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel and Distributed Systems*, 25(11):3025–3035, 2014.
- [109] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng. A survey on zero-knowledge proof in blockchain. *IEEE Network*, 35(4):198–205, 2021.
- [110] J. von zur Gathen. Algebraic complexity theory. *Annual Review of Computer Science*, 3(1):317–348, 1988.
- [111] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 733–764, Cham, 2019. Springer International Publishing.
- [112] S. Yang, A. R. Butt, Y. C. Hu, and S. P. Midkiff. Lightweight monitoring of the progress of remotely executing computations. In E. Ayguadé, G. Baumgartner, J. Ramanujam, and P. Sadayappan, editors, *Languages and Compilers for Parallel Computing*, pages 319–333, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [113] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.
- [114] X. Yu, Z. Yan, and A. V. Vasilakos. A survey of verifiable computation. *Mobile Networks and Applications*, 22:438–453, 2017.

- [115] 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*, 13(7):651–659, 2021.

Appendix A

DiSCuS Experimental Results

An automated benchmarking harness is included in the DiSCuS framework. When initialising the benchmarks, a collection of nodes run the DiSCuS client application, connecting to each other and sharing network topological information. Then, with input from the user, the benchmarks start. For an initial *Work Group* size of 1 the initiating node runs each of the 5 benchmark applications 30 times, averaging the recorded latency. This is then repeated for a *Work Group* 1 node larger, and this process continues until reaching a preconfigured maximum *Work Group* size. Unfortunately, the raw data was discarded with only the average across the 30 runs recorded.

These experimental results were gathered on a single multi-core desktop computer, featuring an Intel i7-9700 CPU clocked at 3.0 GHz, with 8 cores. The benchmarks were bespoke and constructed in the `GlobalInstruction` intermediate language as described in Chapter 6.

In order to normalise these results in relation to each other in the body of the thesis they are often converted into “equivalent plaintext operations per-second”. This exploits SIMD-like behaviours of HE schemes built on lattices with large polynomial ciphertexts. Often the number of SIMD lanes afforded by a HE scheme of this style is equivalent to the ring order of the ciphertexts. Conveniently, the computational complexity of HE addition is linear in the ciphertext ring order, and so by this linear factor can be extracted by multiplying the number of ciphertext operations per-second by the number of SIMD lanes to calculate the number of plaintext operations per-second. To avoid confusion the results in this appendix are presented in their raw form, the benchmark latency in ms.

The ciphertext ring order was set to 16, the plaintext precision is set to 8 bits, and the remainder of the parameters were programmatically generated based on heuristic error bounds and computational depth.

The results of the experiments are listed in Table A.1. The sections to follow describe each experiment in more detail.

A.1 Independent Plaintext Addition

This experiment involved evaluating 1,000 independent plaintext integer addition operations. The equivalent of

```
let y = [u8; 1_000];
let z = [u8; 1_000];
for i in 0..1_000 {
    x[i] = y[i] + z[i];
}
```

Experiment	Independent Plaintext Addition	Average Execution Time (ms)			
		Independent HE Addition	Independent HE Addition in DCRT Form	Dependent HE Addition	Dependent HE Addition in DCRT Form
DiSCuS (Compiled)	24	73	13	73	128
DiSCuS (Interpreted)	2991	126	1057	110	5600
DiSCuS (1 Node)	2907	7455	942	4575	4976
DiSCuS (2 Nodes)	2823	6999	722	4693	5600
DiSCuS (3 Nodes)	2641	4774	743	4943	5072
DiSCuS (4 Nodes)	2489	3801	683	5111	4848
DiSCuS (5 Nodes)	2403	3120	674	5122	4832
DiSCuS (6 Nodes)	2439	2551	669	5355	4960
DiSCuS (7 Nodes)	2343	2322	643	5400	4960
DiSCuS (8 Nodes)	2311	2046	610	5963	4944
DiSCuS (9 Nodes)	2296	1922	613	6292	5024
DiSCuS (10 Nodes)	2250	1766	615	6583	5120
DiSCuS (11 Nodes)	2257	1675	620	6830	4944
DiSCuS (12 Nodes)	2267	1513	633	6942	4960
DiSCuS (13 Nodes)	2333	1410	643	7464	5168
DiSCuS (14 Nodes)	2248	1626	628	7719	5072
DiSCuS (15 Nodes)	2245	1449	624	8196	4848
DiSCuS (16 Nodes)		1560			

Table A.1: Independent plaintext addition experiment results.

```
}
```

With initial values randomly sampled from a domain bounded by the plaintext precision.

A.2 Independent HE Addition

This experiment involved evaluating 100 independent encrypted HE addition operations. The equivalent of

```
let c0 = [CipherText; 100];
let c1 = [CipherText; 100];
for i in 0..100 {
  c[i] = c0[i] + c1[i];
}
```

The ciphertexts were generated by encrypting known random before beginning the benchmark and were handled in their default large polynomial form.

A.3 Independent HE Addition in DCRT Form

As with the previous experiment, this experiment featured 100 independent encrypted HE addition operations. This time the ciphertexts were encoded in DCRT form.

A.4 Dependent HE Addition

This experiment features a sequence of 100 dependent encrypted HE additions

```
let c = CipherText;
let c0 = [CipherText; 100];
for i in 0..100 {
  c = c + c0[i];
}
```

In this experiment ciphertexts were handled in their default large polynomial form.

A.5 Dependent HE Addition in DCRT Form

This experiment features 100 dependent encrypted HE addition operations, but this time ciphertexts are encoded in DCRT form.

Appendix B

DiSCuS Polynomial Regression

Polynomial regression was used to fit a performance curve of the form

$$t = \frac{c}{n} + o_n n + o_i c \quad (\text{B.1})$$

to the results generated by the DiSCuS benchmark experiments. This was important to extract constants o_n and o_i and reason about the contributing factors to the performance of the DiSCuS system. Python was used as a data science language of choice, with the **sklearn** library providing a polynomial regression implementation and **numpy** for importing and handling data.

Polynomial regression requires that the polynomial form that is used to fit the data has no terms with order less than 0. In order to satisfy this constraint the performance equation had to be adjusted to:

$$tn = c + o_i cn + o_n n^2 \quad (\text{B.2})$$

This way, an off-the-shelf polynomial regression implementation will provide terms for c , $o_i c$ and o_n , from which o_i can be easily extracted. This does, however, require that the results are preprocessed with each latency t multiplied by the number of nodes used in that experiment.

First the relevant libraries were imported, then data was read in from a **csv** format. This data was extracted and reordered as required, and then t was multiplied by the number of nodes used in that given benchmark run, n . Following this, an off-the-shelf polynomial regression implementation from **sklearn** was used to fit a curve to the input data. This yielded constants c , $o_i c$ and o_n , and a final step extracted o_i via a division by c . The Python code used is included here:

```
import sklearn
import numpy as np
import seaborn as sns

val = np.loadtxt("../data/he_vector_add.csv", dtype=str, delimiter=",")
print(val)
x = val[:,0][1:].astype('int')
print(x)
y = val[:,2][1:].astype('int')
print(y)

for i in range(16):
```

```

y[i] = y[i] * x[i]

from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=False)
poly_features = poly.fit_transform(x.reshape(-1, 1))

from sklearn.linear_model import LinearRegression
poly_reg_model = LinearRegression()
poly_reg_model.fit(poly_features, y)

c = poly_reg_model.intercept_
o_nc = poly_reg_model.coef_[1]
o_i = poly_reg_model.coef_[0]

o_n = o_nc / c

print(c)
print(o_n)
print(o_i)

```

The exception is when dealing with dependent HE addition experiment where instructions are not encoded in DCRT form. As this experiment has no anticipated parallelism the performance is expected to fit a curve of the form:

$$t = c + o_n n + o_i c \quad (\text{B.3})$$

As this is only a 1st degree polynomial, polynomial regression will only provide a value for $c + o_i c$ and o_n , from which extracting c and o_i is non-trivial and at best requires some educated guess work to predict c .