Zero-knowledge Proofs: Efficient Techniques for Combination Statements and their Applications

by

Chaya Ganesh

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September 2017

Professor Yevgeniy Dodis

Dedication

To all my Gurus, with gratitude.

Acknowledgements

I would like to express my deep sense of gratitude to my advisor, Yevgeniy Dodis, for opening up several opportunities for me. I thank him for his endless support and encouragement. I am especially thankful to him for giving me great freedom and allowing me to discover what I enjoy. His enthusiasm, energy and passion will always inspire me.

I would like to thank all my internship mentors for investing in me: Melissa Chase at Microsoft Research, for introducing me to zero-knowledge; Vlad Kolesnikov at Bell Labs, for believing in me and for being an excellent mentor; Payman Mohassel at Visa Research, for the wonderful time I had working with him, and for invaluable career advice. I am grateful to all of them for giving me the opportunity to work with them. I immensely enjoyed all my internships; they have all been great learning experiences. I am also grateful to my hosts during various research visits: Benny Applebaum at Tel Aviv University for hosting me when I still had a lot to learn; and Arpita Patra at Indian Institute of Science for being both a friend and a collaborator. I thank all my co-authors for the many hours of fun and hard-work.

I am grateful to Victor Shoup for teaching wonderful courses on Cryptography and Number Theory. I thank all my committee members for their constructive comments that helped improve this work: Oded Regev, Victor Shoup, Payman Mohassel and Vlad Kolesnikov.

I thank the staff of Computer Science Department, especially Rosemary Amico and Santiago Pizzini, for all their help and support.

I thank the many friends I made during my stay at Courant: Adriana for her encouragement and support during difficult times; Aris for being patient with all my questions when I was starting out; Azam, Deva, Huck, Igor, Laura, Noah, Omri, Sasha, Sandro, Shravas, Sid, Siyao for many discussions, and their company. I am grateful to all of them for creating a stimulating and encouraging environment, and for many enjoyable conversations, both academic and otherwise. I thank Shiva, Talal and Varun for many delightful dinners, conversations and game nights in their company.

I thank my roommates, Seher and Ashwin, for all the above and so much more: their warmth, and for always being there. I cannot imagine this journey being half the fun without them.

I thank Esha for standing by me through all times; good, and difficult.

Finally, I thank my family for their continuous love and unflinching support.

Abstract

Zero-knowledge proofs provide a powerful tool, which allows a prover to convince a verifier that a statement is true without revealing any further information. It is known that every language in NP has a zero knowledge proof system, thus opening up several cryptographic applications. While true in theory, designing proof systems that are *efficient* to be used in practice remains challenging. The most common and most efficient systems implemented are approaches based on sigma protocols, and approaches based on SNARKs (Succinct Non-interactive ARguments of Knowledge). Each approach has its own advantages and shortcomings, and each is suited for proving certain statements.

While sigma protocols are efficient for algebraic statements, they are expensive for non-algebraic statements. For example, proving statements about hash functions that are expressed as Boolean circuits would entail writing each gate in the circuit as an algebraic relation, resulting in several exponentiations per gate in the circuit.

SNARKs, on the other hand, result in short proofs and efficient verification, and are better suited for proving statements about hash functions. But proving an algebraic statement, for instance, knowledge of discrete logarithm, is expensive as the prover needs to perform public-key operations proportional to the size of the circuit.

Recent works achieve zero-knowledge proofs that are efficient for statements phrased as Boolean circuits based on Garbled circuits (GC). This, again, is expensive for large circuits, in addition to being inherently interactive. Thus, SNARKs, and GC-based approaches are better suited for non-algebraic statements, and sigma protocols are efficient for algebraic statements.

But in some applications, one is interested in proving *combination* statements, that is, statements that have both algebraic and non-algebraic components. For example, consider proving knowledge of x such that $H(g^x) = y$ for a hash function H, and public y. The state of the art fails to take advantage of the best of all worlds and has to forgo the efficiency of one approach to obtain the other's. In this work, we ask how to efficiently prove a statement that is a combination of algebraic and non-algebraic statements.

1. We first show how to combine the GC-based approach with sigma protocols: we give protocols for combination statements where a garbled circuit is used for the Boolean circuit component of the statement and sigma protocol for the algebraic component. We show applications of our protocols in achieving anonymous credentials based on standard signatures.

2. Then, we study how to combine sigma protocol proofs with SNARKs to obtain non-interactive arguments for combination statements. We show appli-

cations of our techniques to privacy-preserving protocols on the blockchain.

3. Finally, we study garbled circuits as a primitive and present an efficient way of hashing garbled circuits. We show applications of our hashing technique, including application to GC-based zero-knowledge.

Contents

Dedication . . . iv
Acknowledgements . . . v
Abstract . . . vi
List of Figures . . . xi
List of Tables . . . xii

1 Introduction **1**
1.1 Overview of Results . . . 1
1.1.1 Zero-knowledge Proofs . . . 2
1.1.2 Garbled Circuits . . . 2
1.2 Zero-knowledge Proofs . . . 2
1.2.1 Our Results . . . 3
1.2.2 Applications . . . 5
1.3 Garbled Circuits . . . 9
1.3.1 Our Results . . . 9
1.3.2 Applications . . . 10
1.4 Roadmap . . . 12

2 Preliminaries **14**
2.1 Notation . . . 14
2.2 Zero-knowledge Proofs . . . 14
2.2.1 Sigma protocols . . . 16
2.2.2 Non-interactive Zero-knowledge Proofs . . . 17
2.3 Garbled Circuits . . . 19
2.3.1 Yao's construction . . . 22
2.3.2 Free-XOR and other optimizations . . . 22
2.4 Garbled Circuits for ZK . . . 23
2.4.1 Oblivious Transfer . . . 23
2.4.2 ZK Proof Based on Garbled Circuits . . . 24
2.5 SNARKs for Arithmetic Circuits . . . 25
2.5.1 Quadratic Arithmetic Programs . . . 25
2.5.2 Bilinear Maps . . . 26

2.5.3 zk-SNARK construction from QAP . . . 27

3 ZK for Combination Statements 29
3.1 Sigma Protocols and GC for Combination Statements . . . 29
3.1.1 Preliminaries . . . 29
3.1.2 Proving Non-algebraic Statements on Algebraic Commitments 30
3.1.3 First Protocol . . . 32
3.1.4 Second Protocol . . . 37
3.1.5 Efficiency Comparison and Optimizations . . . 42
3.2 Sigma Protocols and SNARKs for Combination Statements . . . 44
3.2.1 Proof of equality of committed values . . . 44
3.2.2 SNARK on committed input . . . 45
3.2.3 SNARK on committed input/output . . . 51
3.2.4 Sigma protocols on committed outputs . . . 54

4 Applications of ZK for Combination Statements 55
4.1 Building Blocks for Privacy-Preserving Signature Verification . . . 55
4.1.1 Proving that a committed value is the hash of another committed value . . . 55
4.1.2 Proof of equality of committed values in different groups . . 56
4.1.3 Proof of equality of discrete logarithm of a committed value and another committed value . . . 57
4.2 Privacy-Preserving Signature Verification . . . 60
4.2.1 RSA signatures . . . 60
4.2.2 The DSA Scheme. . . . 63
4.3 Secure computation on committed/signed inputs . . . 67
4.4 Building Blocks for Privacy-Preserving Proof of Solvency . . . 68
4.4.1 Proof of Knowledge of Double Discrete Logarithm in Elliptic Curve Groups . . . 68
4.5 Proof of Solvency . . . 73
4.5.1 Proof of assets . . . 73
4.5.2 Proof of liabilities . . . 74
4.5.3 Privacy-preserving proof of solvency . . . 77

5 Hashing Garbled Circuits for Free 78
5.1 Overview . . . 78
5.1.1 Related Work . . . 82
5.2 GC hashing scheme . . . 83
5.2.1 Hashed Garbled Circuit security . . . 83
5.2.2 Our Construction . . . 84
5.2.3 Hashing in half-gates garbling scheme . . . 90
5.2.4 Performance and Impact . . . 93

5.3 Application to Zero-Knowledge 97

6 Conclusion **101**

Bibliography 102

List of Figures

2.1 The ideal functionality $\mathcal{F}_{COT}$ . . . 24

3.1 The ideal functionality $\mathcal{F}_{\mathsf{Com},f}$ . . . 31
3.2 The Protocol $\Pi_{\mathsf{Com},f}$ . . . 34
3.3 The Protocol $\Pi_{\mathsf{MAC},f}$ . . . 39
3.4 The Protocol comEq . . . 44
3.5 The Protocol comInSnark . . . 48
3.6 The Protocol comIOSnark . . . 54

4.1 The ideal functionality $\mathcal{F}_{Hash}$ . . . 56
4.2 The Protocol Π_{Hash} . . . 56
4.3 The ideal functionality $\mathcal{F}_{Eq}$ . . . 57
4.4 The Protocol Π_{Eq} . . . 57
4.5 Double discrete logarithm proof . . . 58
4.6 pointAddition : $\mathsf{PK}\{(P = (P_x, P_y), Q = (Q_x, Q_y)) : T = (T_x, T_y) = P + Q \wedge \mathsf{C}_1 = \mathsf{Com}_q(P_x) \wedge \mathsf{C}_2 = \mathsf{Com}_q(P_y) \wedge \mathsf{C}_3 = \mathsf{Com}_q(Q_x) \wedge \mathsf{C}_4 = \mathsf{Com}_q(Q_y)\}$ . . . 70
4.7 ec-ddlog : $\mathsf{PK}\{(\lambda, x, y, r, r_1, r_2) : \mathsf{Com}_p(\lambda) = \lambda P + rQ \wedge \mathsf{Com}_q(x) = xP' + r_1Q' \wedge \mathsf{Com}_q(y) = yP' + r_2Q' \wedge (x, y) = \lambda P\}$ . . . 72
4.8 Proof of assets . . . 75
4.9 Proof of liabilities . . . 76
4.10 Proof of solvency . . . 77

5.1 The Free Hash garbling scheme $\mathsf{h}\mathcal{G}$ . . . 86
5.2 The Half-Gate Free Hash garbling scheme $\mathsf{half}\mathcal{G}$ . . . 91
5.3 The GC based Σ-protocol . . . 100

List of Tables

5.1 Free Hash Implementation results 94
5.2 SFE Performance improvement using Free Hash 96
5.3 A billion-gate circuit. Execution time estimates of cut-and-choose with our improvements to achieve cheating probability of 2^{-40} . . . 97
5.4 A billion-gate circuit. Execution time estimates of cut-and-choose with our improvements to achieve deterrence of $\epsilon = 0.9$. 97

Chapter 1

Introduction

Zero-knowledge proofs [GMR85] provide an extremely powerful tool, which allows a prover to convince a verifier that a statement is true without revealing any further information. Since their conception, zero-knowledge proofs have emerged as a fundamental object in modern cryptography, with deep connections to the theory of computation [GMW86, For87, BOGG+90, Vad99].

Zero-knowledge proofs have found countless applications; some of them being identification schemes [FFS87], group signature schemes [CS97b], public-key encryption [NY90], anonymous credentials [CL01], voting [CF85], and secure multi-party computation [GMW87a]. Most recently, zero-knowledge proofs have been used as an important tool in alt-coins like ZCash, Monero, etc. to make the transactions private and anonymous [BCG+14, NMT].

It has been shown that every NP language has a zero knowledge proof system [GMW87b], opening up the possibility for a vast range of privacy preserving applications. However, while this is true in theory, designing proof systems that are efficient enough to be used is significantly more challenging. In reality, we only have a few techniques for efficient proofs, and those only apply to a restricted set of languages.

In this dissertation, we study efficient zero-knowledge proofs for the kind of statements that come up in applications.

1.1 Overview of Results

We design efficient zero-knowledge proof systems to prove *combination* statements: statements that have algebraic and non-algebraic components, that often come up in practice. We focus on the applications of our constructions to anonymous credentials and privacy-preserving Bitcoin audits. We then study Garbled circuits, design an efficient technique to hash them and show applications, including application to zero-knowledge.

1.1.1 Zero-knowledge Proofs

We first give constructions that efficiently combine sigma protocol proofs with zero-knowledge proofs based on garbled circuits. Our constructions have the property that, for a combination statement, the circuit that is garbled for the GC-based proof is independent of the algebraic component of the statement. In particular, expensive group operations are not part of the circuit that is garbled. In addition, the number of public key operations in the sigma proof is independent of the size of the circuit that is garbled. Next, we study efficient non-interactive arguments for combination statements. Using the garbled circuit approach for the non-algebraic component results in a protocol that is inherently interactive. We begin with SNARKs that are efficient for statements represented as Boolean or arithmetic circuits, and show how to use them together with sigma protocols. The prover complexity in SNARKs grows with the size of the circuit. Our constructions have the desired property that algebraic component of the combination statement is not represented as a circuit, and the public-key operations of the sigma protocol is independent of the circuit size of the non-algebraic component.

1.1.2 Garbled Circuits

Next, we improve (actually show how to achieve for free) a core garbling feature of GC, circuit hashing. We propose a definition that is weaker than standard collision resistance, but suffices for certain applications of garbled circuits. We then present constructions that output a garbled circuit and its hash that satisfy our definition of GC hash, at no additional computational overhead. We discuss applications of our GC hashing to GC-based zero-knowledge. We also show how this improves standard GC-based cut-and-choose protocols for two-party computation, and evaluation of private certified functions.

1.2 Zero-knowledge Proofs

Sigma protocols are proof systems that focus on proving algebraic statements, i.e. statements about discrete logarithms, roots, or polynomial relationships between values [Sch90, GQ88, CS97b, GS08]. One could, of course, express any NP relation as a combination of algebraic statements, for example by expressing the relation as a circuit, and expressing each gate as an algebraic relation between input and output wires. But if we were to take this approach to prove a statement using sigma protocols we would need several exponentiations per gate in the circuit. This becomes prohibitively expensive for large circuits (for example a circuit computing a cryptographic hash function or block cipher).

Recently, [JKO13] introduced a new approach for proving statements phrased

as boolean circuits, based on garbled circuits. Their construction has the advantage that it only requires a few symmetric key operations per gate, making it dramatically more efficient than a sigma-protocol-based solution for non-algebraic statements. This means that it is finally practical to prove statements about complex operations such as hash functions or block ciphers. For instance, zero knowledge proofs for an AES circuit or a SHA256 circuit can be done in milliseconds on standard PCs using state of the art implementations for garbled circuits. On the other hand, expressing many public key operations as a circuit is still extremely expensive. Consider, for example, a circuit computing modular exponentiation on a cryptographic group - the result would be much larger than the circuit computing a hash function, and computing a garbled circuit for such a computation would be too expensive to be practical.

Succinct Non-interactive ARguments of Knowledge (SNARKs) [Gro10, GGPR13] allow for very efficient verification and short proofs. They assume an honestly generated common reference string (CRS), and the prover performs public-key operations proportional to the size of the arithmetic circuit representing the statement. Thus, though SNARKs are suited for non-algebraic statements, like a cryptographic hash function, they are not well-suited for algebraic relations due to the large size of the circuit representation of group operations.

Now we have very different techniques for achieving zero knowledge proofs for algebraic and non-algebraic statements. But in some applications, one is interested in proving statements that combine the two. For example, what if we want an efficient protocol for proving knowledge of a DSA or RSA signature, whose verification requires computing both a hash function and several exponentiations?

The state of the art fails to take advantage of the best of all worlds and has to forgo the efficiency of one approach to obtain the other's. One might consider directly combining the protocols, but a naive solution would allow a cheating prover to use a different witness for the algebraic and non-algebraic components of the computation and produce a convincing proof for a statement for which there is no single valid witness. Thus, one of the basic challenges is to bind the values committed to in the sigma protocols to the prover's inputs in the GC-based zero knowledge proof or in the SNARK, without having to perform expensive group operations (e.g. exponentiation) inside the circuit, and without proving large-circuit statements using sigma protocols.

1.2.1 Our Results

We study the problem of combining proof systems for algebraic and non-algebraic statements, and obtain the following results.

1. Sigma protocols and Garbled circuits.

- Given an algebraic commitment C, we propose protocols for proving that C is a commitment to x such that $f(x) = 1$ where f is expressed as a boolean circuit. Both constructions have the desired property that the GC-based component is dominated by the cost of garbling f (i.e. not garbling expensive group operations), and the total number of public-key operations is independent of the size of f.

  More specifically, our first solution has public key operations proportional to the maximum bit length of the input ($|x|$), and symmetric-key operations proportional to the number of gates in f. The second has public-key operations proportional to the statistical security parameter s and symmetric-key operations proportional to the number of gates in f and the length of input.

  Existing solutions either require public-key operations proportional to the size of f, or need to garble circuits for expensive group operations such as exponentiations in large groups.

- Building directly on these protocols, we show how to implement a proof that one committed message is the hash of another, and a proof that two commitments in different groups commit to the same value.

- Finally, we show how we can combine all of these protocols to obtain an efficient proof of knowledge of a signature on a committed message for RSA-FDH, DSA, and EC-DSA signatures. This easily extends to standardized variants of RSA like RSA-PSS.

2. Sigma protocols and SNARKs.

   - Given algebraic commitments C_1 and C_2, we construct NIZKs for proving that C_1 and C_2 are commitments to x and y respectively, such that $f(x) = y$. In particular, we show efficient techniques for proving that the input used in the zk-SNARK statement is the same as the input committed to by the algebraic commitment (similarly for the outputs). This enables efficient switching between the algebraic and arithmetic world.

   - We give an efficient NIZK for proving knowledge of double discrete logarithms over elliptic curve groups. In particular, we show techniques for proving knowledge of x such that C is an algebraic commitment to y, where $y = g^x$ and g is a generator for an elliptic curve group.

   - Building on the above protocols, we construct a privacy-preserving proof of solvency for Bitcoin.

We refer the reader to Chapter 3 for our constructions; Section 3.1 for combining

sigma protocols and garbled circuits, and Section 3.2 for combining sigma protocols and SNARKs.

1.2.2 Applications

We primarily focus on anonymous credentials and proof of solvency here, although we believe our results will be applicable to many other privacy protocols.

Anonymous Credentials. Anonymous credential systems were introduced by Chaum [Cha86]. A credential system allows a user to obtain credentials from an organization and at some later point prove to a verifier (either the same organization or some other party) that she has been given appropriate credentials. More specifically, the user's credentials will contain a set of attributes, and the verifier will require that the user prove that the attributes in her credential satisfy some policy. We say the system is anonymous if this proof does not reveal anything beyond this fact.

There have been several proposals for constructions of anonymous credential systems [CL01, CL04, BCKL08, Bra99, BL13]. In general, they all follow a similar approach: the credential is a signature from the organization on the user's attributes. To prove possession of valid credentials, the user will first commit to her attributes, then prove, in zero knowledge, knowledge of a signature on the committed attributes, and finally prove, again in zero knowledge, that the committed attributes satisfy the policy. To make these zero knowledge proofs efficient, most of the proposed credential systems are based on sigma protocols, which as described above give efficient proofs of knowledge for certain algebraic statements. This in turn means that the signatures used must be specially designed so that a sigma protocol can be used to prove knowledge of a signature on a committed message. (We note that, the protocols of [Bra99, BL13] work slightly differently in that the user and organization jointly compute the proof of knowledge of a signature as part of the credential issuance. However, they still use a customized issuing protocol which would not be compatible with standardized signatures, and they use sigma protocols exactly as described here to prove that the committed attributes satisfy the policy.)

But what if we want to base our credentials on a standard signature such as FDH-RSA or DSA which includes hashing the message? Or what if we want the user to be able to prove a statement about his attributes that is not easily expressible as an algebraic relation? In Chapter 4, we show how to use our constructions to base credentials on standard signatures.

Proof of Solvency. Bitcoin as a digital currency has acheived unprecedented success and deployment. Due to the difficulty in managing cryptographic keys,

many users, in practice use online *exchanges* that manage the keys on behalf of the users. Bitcoin exchanges securely hold bitcoins on their customers' behalf, offer conversion between bitcoin and other currencies, and provide other services similar to online banking. Though convenient, the use of exchanges renders the users vulnerable to loss of their assets. A number of exchanges declared bankruptcy after losing their customers' bitcoin holdings due to a variety of reasons like theft, fraud or technical mistakes. The infamous example of Mt.Gox, which was the oldest and largest exchange, led to loss of over 450MUSD in customer assets. A desirable safeguard of the users against such losses is a demonstration that an exchange controls enough bitcoins to settle the accounts of all of its customers. Specifically, what is desired is for an exchange to prove that it is *solvent*. Though an exchange could demonstrate it is solvent by simply transferring all its assets to a fresh public key, such an approach would reveal confidential information like the addresses controlled by the exchange, the size of business etc which is potentially sensitive for both customers and the exchange. A cryptographic proof of solvency allows an exchange to demonstrate it is solvent while not revealing any other information. A proof of reserves and a corresponding proof of liabilities together constitute a proof of solvency.

A cryptographic proof of liabilities was first proposed by Maxwell [Wil]. However, this solution was not entirely private as it leaked information about the number and size of customer accounts. A proof of solvency that reveals no other information, called Provisions was proposed in [DBB+15]. But, Provisions assumes that the public key corresponding to a Bitcoin address is available on the blockchain, and hence does not enable using addresses where the public keys are unknown. A Bitcoin address is a hash, and only a hash of the public key is visible on the blockchain. Hence, Provisions is not compatible with Bitcoin. A Bitcoin address is a 160-bit hash of the public portion of a public/private ECDSA keypair [bit] where the public portion is derived from the private key by an exponentiation operation on the secp256k1 curve [sec]. Thus a proof of solvency for Bitcoin would have the exchange show that it knows the private keys corresponding to some hashed public keys available on the blockchain.

Thus any proof of solvency for a Bitcoin exchange (or any other cryptocurrency) must deal with a zero-knowledge proof that combines both arithmetic and algebraic statements. In particular, the exchange wants to show that it knows a secret x such that $H(g^x) = y$ where H is a hash function such as SHA-256. The statement has both algebraic (g^x) and Boolean (hash function H) parts. One could express the function composition as a purely algebraic or Boolean function and then use Sigma protocols or zk-SNARKs respectively; but in the former case, the proof size and verification time will be quite large, while in the latter, the proof generation time will increase substantially. Ideally, one would like to use a sigma protocol for the algebraic part and a zk-SNARK for the Boolean part, and then combine the

two proofs while preserving zero-knowledge. In Chapter 4, we show how to use one of our constructions to obtain a privacy-preserving proof of solvency for Bitcoin.

Applications of Our Results.

- **Anonymous Credentials based on RSA, DSA, EC-DSA signatures.** The most direct application in the context of anonymous credentials would be to use RSA, DSA, or EC-DSA signatures directly as credentials but still allow for privacy preserving presentation protocols. This would be slower than existing credential systems, but it would have the advantage that the issuer would not have to perform a complex protocol, but would only have to issue standardized signatures. It further enables interoperability with existing libraries and non-private credential applications. The work of Delignat-Lavaud et al. [DLFKP16] achieve a similar result using only zk-SNARKs. This, as discussed earlier is inefficient for the algebraic component. While our first result gives a solution in the interactive setting our second result gives a non-interactive solution.

  Alternatively, we could construct a service which allows users to convert their non-private credentials (based on RSA/DSA/EC-DSA signatures) into traditional anonymous credentials (e.g. Idemix [ide10] or UProve [PZ13] tokens, or keyed-verification credentials [CMZ14]). Using our new protocol, the service could perform that conversion *without knowing the user's attributes*: the user would commit to his attributes, prove using our protocol that they have been signed, and then obtain from the service an anonymous credential encoding the same attributes. (All of these anonymous credential systems allow for issuing credentials on committed attributes.)

- **Anonymous Credentials with more general policies.** Even if we consider a system based on traditional anonymous credentials, we might use our protocols to allow the user to prove that his attributes satisfy a more complicated policy. For example, he might want to release the hash of one of his attributes and prove that that has been done correctly, or prove that an attribute has been encrypted using a standard encryption scheme like RSA-OAEP.

  Our protocols could also be used to prove that a user's attributes fall in a given range, or to prove statements about comparisons between attributes. If the range of values possible for each attribute is small, we already have reasonably efficient solutions - the user can just commit to each bit of the value, and do a straightforward proof. However this becomes expensive when the range gets larger, in which case the most efficient known approach is based on integer commitments [FO97] and requires several exponentiations with

an RSA modulus where the exponent is larger than the group order (e.g. a roughly 2000 bit exponentiation with a 2000 bit modulus for reasonable security parameters). Alternatively we can use our second scheme, which only requires a number of public-key operations linear in the security parameter (e.g. 60), and allows those operations to use much more efficient elliptic curve groups.

- **Converting between different commitment schemes.** There are many protocols based around commitments, and ideally we would be able to combine these protocols arbitrarily. For example, if we have an efficient protocol for proving that a committed tag matches one of the attributes in a user's credential, and another protocol for proving that a committed tag is not on a list of revoked values, then we would be able to combine the two protocols to prove that the user's credential has not been revoked. However, often the protocols will be based on different commitment schemes, or even worse, on schemes that operate in different sized groups. (For example UProve credentials can be instantiated in standardized elliptic curve groups like those used for EC-DSA, while revocation systems like that in [Ngu05] require pairing groups; to combine the two we would need to find a pairing group whose group order matches one of the standardized curves. Finding a pairing group to match a specific group order often incurs a significant cost in efficiency.) With our protocol for converting between commitment schemes we could choose the most efficient groups for each, and then the user would merely prove that he has used the same attributes in each. Before our work, the only known approach to convert between groups of different sizes was to use integer commitments, which as described above can be quite expensive.

- **2PC with authenticated input.** As input to a secure computation protocol, sometimes it is desirable to use previously committed [JS07] or signed [CZ09] inputs. In our constructions, we show how to commit to an input x and prove knowledge of x (or prove knowledge of a signature on x) and a non-algebraic statement $f(x) = 1$ using garbled circuits. As we discuss in Section 4.3, it is relatively easy to extend our construction to also allow secure two-party computation of $g(x, y)$ where x is the prover's input and y the verifier's, hence obtaining secure two-party computation on signed/committed inputs. The benefit of this approach is that checking the signature takes place outside the secure two-party computation and can be significantly more efficient.

- **Other privacy-preserving protocols.** Converting between commitment schemes, comparing committed values, or proving other non-algebraic statements come up in many other privacy/anonymity scenarios. Our techniques

for composing proofs could be useful in reducing the size of CRS in applications such as the anonymous decentralized digital cryptocurrencies. ZCash, for example, uses zk-SNARKs to prove a massive statement containing many different smaller components. For example, at a high level, one of the statements being proven in ZCash is of the form: I have knowledge of x_i's such that $H(x_1||H(x_2||\ldots H(x_n))) = y$ for a large value of n. The CRS generated for proving this statement is extremely large (in the gigabytes for ZCash) and cannot be reused to prove any different statement. A better alternative is to generate a much smaller CRS for proving a statement of the form: I have knowledge of x, y such that $H(x||H(y))$, combined with a technique for composing many such proofs. More generally, one can envision a general system with CRSs for small size statements $C_1, \ldots, C_n$ that enables NIZKs for arbitrary composition of these statements without having to generate new CRSs for each new composition.

1.3 Garbled Circuits

Today Garbled Circuit (GC) is one of the main techniques for secure computation. It has advantages of high performance, low round complexity/low latency, and, importantly, relative engineering simplicity. Both core GC (garbling), as well as the protocols that use garbling, such as Cut-and-Choose (C&C), have been thoroughly investigated and are today highly optimized. Particularly in the semi-honest model there have been quite a number of asymptotic/qualitative improvements since the original protocols of Yao [Yao86] and Goldreich et al. [GMW87a]. Possibly the most important development in the area of practical SFE since the 1980s was the very efficient oblivious transfer (OT) extension technique of Ishai et al. [IKNP03]. This allowed the running of an arbitrarily large number of OTs by executing a small (security parameter) number of (possibly inefficient) "bootstrapping" OT instances and a number of symmetric key primitives. The cheap OTs made a dramatic difference for securely computing functions with large inputs relative to the size of the function, as well as for GMW-like approaches, where OTs are performed in each level of the circuit. Another important GC core improvement is the Free-XOR algorithm [KS08a], which allowed for the evaluation of all XOR gates of a circuit without any computational or communication costs. As SFE moves from theory to practice, even "small" factor improvements can have a significant effect.

1.3.1 Our Results

In this work, we introduce *Free Hash*, a new approach to generating GC hash at no extra cost during GC generation. GC hashing is at the core of the cut-and-

choose technique of GC-based secure function evaluation (SFE). Our main idea is to intertwine hash generation/verification with GC generation and evaluation. While we *allow* an adversary to generate a GC $\widehat{\mathsf{GC}}$ whose hash collides with an honestly generated GC, such a $\widehat{\mathsf{GC}}$ w.h.p. will fail evaluation and cheating will be discovered. Our GC hash is simply a (slightly modified) XOR of all the gate table rows of the GC. It is compatible with Free XOR and half-gates garbling, and can be made to work with many cut-and-choose SFE protocols.

In Chapter 5, we introduce our proposed definition of GC hash security. Our definition is weaker than the standard hash collision guarantees, yet it is possible to make free hashing work with several standard GC constructions. We then present hashed garbling algorithms for standard garbling (based on Just Garble of [BHKR13]) as well as for half-gates garbling of [ZRE15]. We discuss the impact of Free Hash garbling and cut-and-choose protocols. We report on our implementation and its performance evaluation, and discuss the application to certified circuits. We propose a unified cost metric (time) and show higher speeds/smaller computation and communication for the same error probability. We estimate total execution time reduction of about 43% for the cut-and-choose components of [LP11], and of about 64% for [AO12, KM15] in settings we consider (1Gbps channel and hardware AES). We then consider the standard cut-and-choose approach for the special case of zero-knowledge, and discuss application of Free Hash to the GC-based sigma protocol.

1.3.2 Applications

GC hashing is an essential tool for C&C and is employed in many uses of C&C . We start with describing C&C at the high level.

Cut-and-Choose Protocol. C&C was first mentioned in the protocol of Rabin [Rab77] where this concept was used to convince a party that the other party sent it a specially formed integer n. The expression "cut and choose" was introduced later by Chaum in [BCC88] in analogy to a popular cake-sharing problem: given a cake to be divided among two distrustful players, one of them cuts the cake in two shares, and lets the other one choose.

Recall, the basic GC protocol is not secure against cheating GC generator, who can submit a maliciously garbled circuit. Today, C&C is the standard tool in achieving malicious security in secure computation. At the high level, it proceeds as follows. GC generator generates a number of garbled circuits $\mathsf{GC}_1, ..., \mathsf{GC}_n$ and sends them to GC evaluator, who chooses a subset of them (say, half) at random to be opened (with the help of the generator) and verifies the correctness of circuit construction. If all circuits were constructed correctly, the players proceed to securely evaluate the unopened circuits, and take the majority output. It is easy to

see that the probability of the GC generator succeeding in submitting a maliciously garbled circuit is exponentially small in n. We note that significant improvement in the concrete values of n required for a specific probability guarantee was achieved by relatively recent C&C techniques [LP11, Lin13, HKE13, Bra13, LR14, HKK+14, AO12, KM15].

Using GC hashing for C&C. What motivates our work is the following natural idea, which was first formalized in Goyal et al. [GMS08]. To save on communication (usually a more scarce resource than computation), GC generator, firstly, generates all the circuits $\mathsf{GC}_1, ..., \mathsf{GC}_n$ from PRG seeds $s_1, ..., s_n$. Then, instead of sending the circuits $\mathsf{GC}_1, ..., \mathsf{GC}_n$, it sends their hashes $H(\mathsf{GC}_1), ..., H(\mathsf{GC}_n)$. Finally, while the evaluation circuits will need to be sent in full over the network, only the seeds $s_1, ..., s_n$ need to be sent to verify that the GC generator did not cheat in the generation of the opened circuits, saving a significant amount of communication at the cost of computing and checking $H(\mathsf{GC}_i)$ for all n circuits.

On many of today's computing architectures (e.g.Intel PC CPUs, with or without hardware AES), the cost of hashing the GC can be up to 6× greater than the cost of fixed-key garbling. At the same time, today's network speeds are comparable in throughput with hardware-assisted fixed-key garbling (see our calculations in Section 5.2.4).

Hence, eliminating the GC hashing cost will improve SFE performance by eliminating the (smaller of the) cost of hashing or sending the open circuits. We stress that the use of our Free Hash requires syntactic changes in C&C protocols and it provides a security guarantee somewhat distinct from collision-resistant hash. Hence its use in C&C protocols should be evaluated for security. We discuss this in Section 5.2.4.

Additionally, we show that a new computation/communication cost ratio offered by our free GC hash will allow for reduced communication, computation, and execution time, while achieving the same cheating probability.

SFE of private certified functions. One advantage offered by GC is the hiding of the evaluated function from the evaluator. To be more precise, the circuit topology of the function is revealed, but this information leakage can be removed or mitigated by using techniques such as universal circuit [Val76, KS08c, LMS16, KS16] or circuit branch overlay [KKW16].

In practical scenarios, evaluated functions are to be selected as allowed by a mutually agreed policy, e.g., to prevent evaluation of the identity function outputting player's private input. Then evaluating a hidden function presumes either a semi-honest GC generator, or employing a method for preventing/deterring out-of-policy GC generation. An efficient C&C approach does not seem to help prevent cheating here, since check circuits will reveal the evaluated function and will not

be acceptable to the GC generator. Further, depending on policy/application, the zero-knowledge proofs of correctly constructing the circuits may be very expensive.

In many scenarios, Certificate Authorities (CA) may be used to certify the correct generation of GCs. Indeed, this is quite feasible at small to medium scale. Our motivating application here is the private attribute-based credential (ABC) checking. Our results on combining sigma protocol and garbled circuits show that credentials can be based on GCs. Recent concurrent work [KKL+16] also use GCs to build ABCs. While [KKL+16] discuss public policy only, GC-based constructions will not preclude achieving private policy. We note that this is a novel property in the ABC literature, where all previous work (in addition to supporting very small policies only) relied in an essential manner on the policy being known to both prover and verifier.

At the high level, the architecture/steps for evaluation of private CA-certified functions is as follows.

1. CA generates seeds $s_1, ...s_n$ and, for $i = 1, ...n$, CA generates GCs GC_i, GC hashes $H(\mathsf{GC}_i)$ and signatures $\sigma_i = \mathsf{Sign}_{CA}(H(\mathsf{GC}_i))$. It sends all $s_i, H(\mathsf{GC}_i)$, σ_i to ABC verifier V.

2. Prover P and V proceed with execution of the ABC protocols, with the following modification:

   (a) Whenever GC GC_i needs to be sent by V, instead V generates GC_i from s_i and sends to P the pair $(\mathsf{GC}_i, \sigma_i)$.

   (b) P computes $H(\mathsf{GC})$ and verifies the signature σ_i prior to continuing. If the verification or GC evaluation fails, P outputs `abort`.

Free Hash will allow to significantly (up to factor 6) reduce the computational effort required by the CA to support such an application. Indeed the cost of the signature generation can be small and ignored in cases where the signed circuits are large, or a single signature can certify a number of circuits. The latter would be the case where two parties may be expected to evaluate a number of circuits.

1.4 Roadmap

In Chapter 2, we present preliminaries and definitions of technical tools used in the rest of the chapters. Chapter 3 presents constructions of zero-knowledge proofs for combination statements. In Chapter 4, we present more building blocks, and put together our constructions to build credentials based on standard signatures, and privacy-preserving proof of solvency. In Chapter 5, we give our definition and construction of Garbled circuit hashing and discuss application to zero-knowledge.

The results in this dissertation are based on works that appeared in [CGM16], [FGK17], and on parts of another work under submission [AGM17].

Chapter 2

Preliminaries

2.1 Notation

Let PPT denote probabilistic polynomial time. We let κ be the security parameter and $[1, n]$ denote the set $\{1, ..., n\}$. A function is negligible if for all large enough values of the input, it is smaller than the inverse of any polynomial, that is, $f(\cdot)$ is negligible if $\forall c \in \mathbb{N}$, there exists $n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$, it holds that $f(n) < n^{-c}$. We use negl to denote a negligible function. Let S be an infinite set and $X = \{X_s\}_{s \in S}, Y = \{Y_s\}_{s \in S}$ be distribution ensembles. We say X and Y are computationally indistinguishable, if for any PPT distinguisher $\mathcal{D}$ and all sufficiently large $s \in S$, we have $|\Pr[\mathcal{D}(X_s) = 1] - \Pr[\mathcal{D}(Y_s) = 1]| < 1/p(|s|)$ for every polynomial $p(\cdot)$. We denote the i-th bit of a string $\boldsymbol{s}$ by $\boldsymbol{s}[i]$, and use $||$ to denote concatenation of bit strings. We write $x \xleftarrow{R} \mathcal{X}$ to mean sampling a value x uniformly from the set $\mathcal{X}$. For a bit string $\boldsymbol{s}$, we let $\boldsymbol{s}^{\ll i}$ denote the bit string obtained by shifting $\boldsymbol{s}$ by i bits to the left. Throughout, by shift we mean a *circular* shift, where the vacant bit positions are filled not by zeros but by the shifted bits. $\mathsf{lsb}(\boldsymbol{s})$ denotes the least significant bit of string $\boldsymbol{s}$.

2.2 Zero-knowledge Proofs

A zero-knowledge (ZK) proof allows a prover to convince a verifier of the validity of a statement, without revealing any other information. Let R be an efficiently computable binary relation which consists of pairs of the form (x, w) where x is a statement and w is a witness. Let $\mathcal{L}$ be the language associated with the NP relation R: $\mathcal{L} = \{x \mid \exists w : R(x, w) = 1\}$. A zero-knowledge proof for $\mathcal{L}$ lets the prover convince a verifier that $x \in \mathcal{L}$ for a common input x. A proof of knowledge captures not only the truth of a statement $x \in \mathcal{L}$, but also that the prover "possesses" a witness w to this fact. A proof of knowledge for a relation

$R(\cdot,\cdot)$ is an interactive protocol where a prover P convinces a verifier V that P knows a w such that $R(x,w)=1$, where x is a common input to P and V. The prover can always successfully convince the verifier if indeed P knows such a w. Conversely, if P can convince the verifier with reasonably high probability, then it "knows" such a w, that is, such a w can be efficiently computed given x and the code of P. The formal definition follows. In the following, we denote an interactive protocol between P and V by $\langle P,V\rangle$. $\langle P(x),V(y)\rangle$ denotes a protocol where P has input x and V has input y. We denote by $view_V$, the "view" of the verifier in the interaction, consisting of its input x, its random coins, and the sequence of the prover's messages.

Definition 2.2.1 (ZK proof of knowledge)**.** *An interactive protocol $\langle P,V\rangle$ is a zero-knowledge proof of knowledge for an NP relation R if the following properties are satisfied.*

1. *Completeness: For all x,w such that $R(x,w)=1$,*

$$\Pr[\langle P(x,w),V(x)\rangle=1]=1$$

2. *Proof of Knowledge: For every polynomial time prover strategy P^*, there exists an oracle PPT machine K called the extractor such that $K^{P^*}(x)$ outputs w' and*

$$\Pr[\langle P^*(x,w),V(x)\rangle=1\wedge R(x,w')=0]$$

   is negligible in κ.

3. *Zero-knowledge: For every polynomial time verifier V^*, there is a PPT algorithm $\mathcal{S}$ called the simulator such that for every $x\in\mathcal{L}$, and w such that $R(x,w)=1$, the following two distributions are indistinguishable:*

   - $view_{V^*}(\langle P(x,w),V^*(x)\rangle)$
   - $\mathcal{S}(x)$

Honest-verifier zero-knowledge: An interactive proof system $\langle P,V\rangle$ for a language $\mathcal{L}$ is said to be honest-verifier zero knowledge if there exists a PPT algorithm $\mathcal{S}$ called the simulator such that for all $x\in\mathcal{L}$, $view_V(\langle P(x,w),V(x)\rangle)$ and $\mathcal{S}(x)$ are indistinguishable. This definition says that the verifier gains no knowledge from the interaction, as long as it runs the prescribed algorithm V. If the verifier tries to gain some knowledge from its interaction with the prover by deviating from the prescribed protocol, we should consider an arbitrary (but efficient) cheating verifier V^* as in property 3 of the above definition, which is full zero-knowledge.

2.2.1 Sigma protocols

We introduce the notion of a sigma protocol by motivating through an example. Let p and q be primes such that $q|(p-1)$. Let g be an element of order q in $\mathbb{Z}_p^*$. Now suppose that a prover P chooses a random $x \leftarrow \mathbb{Z}_q$, and publishes $y = g^x \mod p$. A verifier V who receives (p, q, x, y) may verify that p, q are prime, g has order q. The following protocol by Schnorr [Sch91] allows P to convince V that he "knows" the unique $x \in \mathbb{Z}_q$ such that $y = g^x \mod p$.

- Input: The prover and the verifier have (p, q, g, y) and the prover additionally has $x \in \mathbb{Z}_q$ such that $y = g^x \mod p$.
- The prover P chooses a random $w \in \mathbb{Z}_q$, and sends $a = g^w \mod p$ to V.
- The verifier V chooses a random *challenge* $r \leftarrow \{0,1\}^\kappa$ and sends to P, for a fixed κ such that $2^\kappa < q$.
- The prover P responds with $e = w + rx \mod q$ to V. The verifier V checks that p, q are prime, g, y have order q, that $g^e = ay^r \mod p$, and accepts if and only if all the above hold.

Intuitively, if some prover P^*, after having sent a, can answer two *different* challenges r and r' correctly, then this means that it could produce e, e' such that $g^e = ay^r \mod p$ and $g^{e'} = ay^{r'} \mod p$. Dividing the two equations, one gets $g^{e-e'} = y^{r-r'} \mod p$. Since by assumption $r \neq r'$, we have $r - r' \neq 0 \mod q$, and hence $(r-r')^{-1}$ exists modulo q. Since the prover knows r, r', e, e', it could have computed $x = g^{(e-e')(r-r')^{-1}} \mod p$. The prover thus *knows* the discrete logarithm, except with probability $2^{-\kappa}$ which is the probability that it answers only one challenge correctly. We also note that the view of an honest verifier can be simulated. The name of the sigma protocol comes from the letter Σ that depicts a protocol with a three-move interaction. We refer the interested reader to the paper by Ivan Damgård [Dam] for an excellent survey on sigma protocols. We now give a formal description below.

Sigma protocols are three round public-coin protocols and are honest-verifier zero-knowledge proof systems. Let R be a relation, and x the common input. The prover's first message is denoted by $a = \mathsf{P}(x)$. The verifier's message is a random string $r \in \{0,1\}^\kappa$. The prover's second message is $e = \mathsf{P}(x, a, r, e)$. The triple (a, r, e) is called a transcript, and if the verifier accepts, that is $\mathsf{V}(x, a, r, e) = 1$, then the transcript is $\mathsf{accepting}$ for x.

Definition 2.2.2 (Σ protocol)**.** *A protocol π is a Σ protocol for a relation R if the following properties are satisfied:*

1. *π is a three round public coin protocol.*
2. *Completeness: If P and V follow the protocol on common input x, and private input w to P such that $R(x, w) = 1$, then $\Pr[\langle \mathsf{P}(x, w), \mathsf{V}(x) \rangle = 1] = 1$.*
3. *Special soundness: There exists a polynomial time algorithm, called the extractor, that, given x and two transcripts $(a, r, e), (a, r', e')$ that are accepting for x, with $r \neq r'$ outputs w' such that $R(x, w') = 1$.*
4. *Special honest verifier zero knowledge: There exists a* PPT *simulator* Sim *such that*
$$\{\mathsf{Sim}(x, r)\}_{x \in \mathcal{L}, r \in \{0,1\}^\kappa} \equiv \{\langle \mathsf{P}(x, w), \mathsf{V}(x, r) \rangle\}_{x \in \mathcal{L}, r \in \{0,1\}^\kappa}$$
where $\mathsf{Sim}(x, r)$ denotes the output of the simulator Sim upon input x and r, and $\langle \mathsf{P}(x, w), \mathsf{V}(x, r) \rangle$ denotes the output transcript of an execution between P and V, where P has input (x, w), V has input x, and r is the challenge determined by V's random tape.

2.2.2 Non-interactive Zero-knowledge Proofs

A model that assumes a trusted setup phase, where a string of a certain structure, also called the public parameters of the system is generated, is called the common reference string (CRS) model. NIZKs in the CRS model were introduced in [BFM88].

Definition 2.2.3 (Non-interactive Zero-knowledge Argument)**.** *A non-interactive zero-knowledge argument for a binary relation R consists of a triple of polynomial time algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *defined as follows.*

- $\mathsf{Setup}(1^\kappa)$ *takes a security parameter κ and outputs a common reference string σ.*
- $\mathsf{Prove}(\sigma, x, w)$ *takes as input the CRS σ, a statement x, and a witness w, and outputs an argument π.*
- $\mathsf{Verify}(\sigma, x, \pi)$ *takes as input the CRS σ, a statement x, and a proof π, and outputs either 1 accepting the argument or 0 rejecting it.*

The algorithms above should satisfy the following properties.

1. *Completeness. For any* $(x, w) \in R$,

$$\Pr\left(\mathsf{Verify}(\sigma, x, \pi) = 1 \ : \ \begin{matrix} \sigma \leftarrow \mathsf{Setup}(1^\kappa) \\ \pi \leftarrow \mathsf{Prove}(\sigma, x, w) \end{matrix}\right) = 1$$

2. *Computational soundness. For all probabilistic polynomial time (*PPT*) adversaries* $\mathcal{A}$*, the following probability is negligible in* κ*:*

$$\Pr\left(\begin{matrix} \mathsf{Verify}(\sigma, \tilde{x}, \tilde{\pi}) = 1 \\ \wedge \tilde{x} \notin L \end{matrix} \ : \ \begin{matrix} \sigma \leftarrow \mathsf{Setup}(1^\kappa) \\ (\tilde{x}, \tilde{\pi}) \leftarrow \mathcal{A}(1^\kappa, \sigma) \end{matrix}\right)$$

3. *Zero-knowledge. There exists a* PPT *simulator* $(\mathcal{S}_1, \mathcal{S}_2)$ *such such that* $\mathcal{S}_1$ *outputs a simulated CRS* σ *and trapdoor* τ*,* $\mathcal{S}_2$ *takes as input* σ*, a statement* x *and* τ *and outputs a simulated proof* π*. Formally, for all* PPT *adversaries* $(\mathcal{A}_1, \mathcal{A}_2)$*, the following is negligible in* κ*.*

$$\left|\Pr\left(\begin{matrix} (x, w) \in R \\ \wedge \mathcal{A}_2(\pi, \mathsf{state}) = 1 \end{matrix} \ : \ \begin{matrix} \sigma \leftarrow \mathsf{Setup}(1^\kappa) \\ (x, w, \mathsf{state}) \leftarrow \mathcal{A}_1(1^\kappa, \sigma) \\ \pi \leftarrow \mathsf{Prove}(\sigma, x, w) \end{matrix}\right) - \Pr\left(\begin{matrix} (x, w) \in R \\ \wedge \mathcal{A}_2(\pi, \mathsf{state}) = 1 \end{matrix} \ : \ \begin{matrix} (\sigma, \tau) \leftarrow \mathcal{S}_1(1^\kappa) \\ (x, w, \mathsf{state}) \leftarrow \mathcal{A}_1(1^\kappa, \sigma) \\ \pi \leftarrow \mathcal{S}_2(\sigma, \tau, x) \end{matrix}\right)\right|$$

Definition 2.2.4 (Non-interactive Zero-knowledge Argument of Knowledge)**.** *A non-interactive zero-knowledge argument of knowledge for a relation* R *is a non-interactive zero-knowledge argument for* R *with the following additional extractability property:*

- *Extraction. For any* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *algorithm* Ext *such that the following probability is negligible in* κ*:*

$$\Pr\left(\begin{matrix} \mathsf{Verify}(\sigma, \tilde{x}, \tilde{\pi}) = 1 \wedge \\ R(\tilde{x}, w') = 0 \end{matrix} \ : \ \begin{matrix} \sigma \leftarrow \mathsf{Setup}(1^\kappa) \\ (\tilde{x}, \tilde{\pi}) \leftarrow \mathcal{A}(1^\kappa, \sigma) \\ w' = \mathsf{Ext}(\tilde{x}, \tilde{\pi}) \end{matrix}\right)$$

Definition 2.2.5 (Zero-knowledge Succinct Non-interactive Argument of Knowledge (zk-SNARK))**.** *A zk-SNARK for a relation* R *is a non-interactive zero-knowledge argument of knowledge for* R *with the following additional property:*

- *Succinctness. For any* x *and* w*, the length of the proof* π *is given by* $|\pi| = \mathsf{poly}(\kappa) \cdot \mathsf{polylog}(|x| + |w|)$.

Sigma protocols and NIZK. It is possible to efficiently compile a Σ protocol (which is honest-verifier ZK) into a non-interactive zero-knowledge proof of knowledge. The Fiat-Shamir transform [FS87] is a way of transforming any public coin zero-knowledge proof into a non-interactive zero-knowledge proof of knowledge. At a high level, the transform works by having the prover compute the verifier's message which is a random challenge by applying an appropriate hash function to the prover's first message. This can be proven secure when the hash function is modeled as a random oracle. The Fiat-Shamir transform removes interaction *and* guarantees zero-knowledge against malicious verifiers (Sigma protocols are only honest-verifier zero-knowledge). Transformations in the CRS model [Dam00, Lin15] are also known. The transformation of [Dam00] gives a 3-round concurrent zero-knowledge protocol in the CRS model, whereas [Lin15] is non-interactive.

Efficient zero knowledge proofs are known which are based on sigma protocols. There exist sigma protocols for various tasks like proving knowledge of discrete logarithm of a value, that a tuple is of the Diffie-Hellman type etc., and it is also possible to efficiently combine sigma protocols to prove compound statements.

In the constructions and protocols presented in further chapters, we make use of zero knowledge proofs of knowledge of discrete logarithms and relations between discrete logarithms. We use the following notation:

$$\mathsf{PK}\{(x, y, \cdots) : \textit{ statements} \text{ about } x, y, \cdots\}$$

In the above, $x, y, \cdots$ are secrets (discrete logarithms), the prover asserts knowledge of $x, y, \cdots$, and that they satisfy *statements*. The other values in the protocol are public.

2.3 Garbled Circuits

Garbled circuits which was introduced by Yao [Yao86] as a tool for secure two party computation, is now a primitive in its own right with many applications. We use the abstraction of garbling schemes [BHR12] introduced by Bellare et al. At a high-level, a garbling scheme consists of the following algorithms: Gb takes a circuit as input and outputs a garbled circuit, encoding information e, and decoding information d. En takes an input x and encoding information and outputs a garbled input X. Eval takes a garbled circuit and garbled input X and outputs a garbled output Y. Finally, De takes a garbled output Y and decoding information and outputs a plain circuit-output (or an error $\perp$).

We note that this deviates from the definition of [BHR12], in that we include $\perp$ in the range of the decoding algorithm De, so it now outputs a plain output value corresponding to a garbled output value or $\perp$ if the garbled output value is invalid. In [JKO13], the authors add an additional verification algorithm Ve to

the garbling scheme. Formally, we define a *verifiable garbling scheme* by a tuple of functions $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Eval}, \mathsf{De}, \mathsf{Ve})$ with each function defined as follows.

- *Garbling* algorithm $\mathsf{Gb}(1^\kappa, \mathcal{C})$: A randomized algorithm which takes as input the security parameter and a circuit $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$ and outputs a tuple of strings $(\mathsf{GC}, \{X_j^0, X_j^1\}_{j\in[n]}, \{Z_j^0, Z_j^1\}_{j\in[m]})$, where GC is the garbled circuit, the values $\{X_j^0, X_j^1\}_{j\in[n]}$ are called the input-wire labels, and the values $\{Z_j^0, Z_j^1\}_{j\in[m]}$ are called the output-wire labels.

- *Encode* algorithm $\mathsf{En}(x, \{X_j^0, X_j^1\}_{j\in[n]})$: a deterministic algorithm that outputs the input wire labels $\mathbf{X} = \{X_i^{\boldsymbol{x}[i]}\}_{i\in[n]}$ corresponding to input x.

- *Evaluation* algorithm $\mathsf{Eval}(\mathsf{GC}, \{X_j\}_{j\in[n]})$: A deterministic algorithm which evaluates garbled circuit GC on input-wire labels $\{X_j\}_{j\in[n]}$, and outputs a garbled output $\mathbf{Y}$.

- *Decode* algorithm $\mathsf{De}(\mathbf{Y}, \{Z_j^0, Z_j^1\}_{j\in[m]})$: A deterministic algorithm that outputs the plaintext output corresponding to $\mathbf{Y}$ or $\perp$ signifying an error if the garbled output $\mathbf{Y}$ is invalid.

- *Verification* algorithm $\mathsf{Ve}(\mathcal{C}, \mathsf{GC}, \{Z_j^0, Z_j^1\}_{j\in[m]}, \{X_j^0, X_j^1\}_{j\in[n]})$: A deterministic algorithm which takes as input a circuit $\mathcal{C}$, garbled circuit GC, input-wire labels $\{X_j^0, X_j^1\}_{j\in[n]}$, and output-wire labels $\{Z_j^0, Z_j^1\}_{j\in[m]}$ and outputs accept if GC is a valid garbling of $\mathcal{C}$ and reject otherwise.

For purposes of brevity, we sometimes write e to mean the encoding information $\{X_j^0, X_j^1\}_{j\in[n]}$, and d to represent the decoding information $\{Z_j^0, Z_j^1\}_{j\in[m]}$. Throughout, we will only be concerned with a class of garbling schemes referred to as *projective* [BHR12], where, when garbling a circuit $\mathcal{C} : \{0,1\}^n \mapsto \{0,1\}^m$, the scheme produces encoding information of the form $e = \left(X_j^0, X_j^1\right)_{j\in[n]}$. The encoded input $\mathbf{X}$ corresponding to $x = (x_j)_{j\in[n]}$ is interpreted as $\mathbf{X} = \mathsf{En}(x, e) = \left(X_j^{x_j}\right)_{j\in[n]}$.

A verifiable garbling scheme may satisfy several properties such as *correctness, privacy, obliviousness, authenticity and verifiability.* We now review some of these notions: (1) *correctness*, (2) *privacy* (3) *authenticity*, and (4) *verifiability.* The definitions for correctness and authenticity are standard: correctness enforces that a correctly garbled circuit, when evaluated, outputs the correct output of the underlying circuit; privacy aims to protect the privacy of encoded inputs; authenticity enforces that the evaluator can only learn the output label that corresponds to the value of the function. *Verifiability* [JKO13] allows one to check that the garbled circuit indeed implements the specified plaintext circuit $\mathcal{C}$.

We include the definitions of these properties.

Definition 2.3.1. *(Correctness) A garbling scheme $\mathcal{G}$ is* correct *if for all input lengths $n \leqslant \mathsf{poly}(\kappa)$, circuits $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$ and inputs $x \in \{0,1\}^n$, the following probability is negligible in κ:*

$$\Pr(\mathsf{De}(\mathsf{Eval}(\mathsf{GC}, \mathsf{En}(e,x)), d) \neq \mathcal{C}(x) : (\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C}))$$

Definition 2.3.2. *(Privacy) A garbling scheme $\mathcal{G}$ has* privacy *if for all input lengths $n \leqslant \mathsf{poly}(\kappa)$, circuits $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$, there exists a* PPT *simulator* Sim *such that for all inputs $x \in \{0,1\}^n$, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the following two distributions are computationally indistinguishable:*

- $\textsc{Real}(\mathcal{C}, x)$ *: run* $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})$*, and output* $(\mathsf{GC}, \mathsf{En}(x, e), d)$*.*
- $\textsc{Ideal}_{\mathsf{Sim}}(\mathcal{C}, \mathcal{C}(x))$*: output* $\mathsf{Sim}(1^\kappa, \mathcal{C}, \mathcal{C}(x))$

Definition 2.3.3. *(Authenticity) A garbling scheme $\mathcal{G}$ is* authentic *if for all input lengths $n \leqslant \mathsf{poly}(\kappa)$, circuits $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$, inputs $x \in \{0,1\}^n$, and all probabilistic polynomial-time adversaries $\mathcal{A}$, the following probability is negligible in κ:*

$$\Pr\left(\begin{array}{c}\widehat{Y} \neq \mathsf{Eval}(\mathsf{GC}, \mathsf{En}(x,e)) \\ \wedge \mathsf{De}(\widehat{Y}, d) \neq \bot\end{array} : \begin{array}{c}(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C}) \\ \widehat{Y} \leftarrow \mathcal{A}(\mathcal{C}, x, \mathsf{GC}, \mathsf{En}(x,e))\end{array}\right)$$

Definition 2.3.4. *(Verifiability) A garbling scheme $\mathcal{G}$ is* verifiable *if for all input lengths $n \leqslant \mathsf{poly}(\kappa)$, circuits $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$, inputs $x \in \{0,1\}^n$, and all probabilistic polynomial-time adversaries $\mathcal{A}$, the following probability is negligible in κ:*

$$\Pr\left(\mathsf{De}(\mathsf{Eval}(\mathsf{GC}, \mathsf{En}(x,e)), d) \neq \mathcal{C}(x) \ : \ \begin{array}{c}(\mathsf{GC}, e, d) \leftarrow \mathcal{A}(1^\kappa, \mathcal{C}) \\ \mathsf{Ve}\,(\mathcal{C}, \mathsf{GC}, d, e) = \mathsf{accept}\end{array}\right)$$

In the definition of verifiability above, we give the decoding information explicitly to the verification algorithm since in some of our constructions, the garbled circuit includes only the garbled tables and not the decoding information. Alternatively, if the decoding information is also part of the garbled circuit itself, we can have the verification algorithm take only the plain circuit, purported garbled circuit and encoding information, and output accept or reject. In some cases, in addition to the verifiability defined above, we require an additional extraction property. We consider circuits with a single bit output below.

Definition 2.3.5. *(Verifiability with extraction) A garbling scheme $\mathcal{G}$ is* verifiable with extraction *if for all input lengths $n \leqslant \mathsf{poly}(\kappa)$, circuits $\mathcal{C} : \{0,1\}^n \to \{0,1\}$, and all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a* PPT *algorithm* Ve_1*, such that, for all x satisfying $\mathcal{C}(x) = 1$, the following probability is negligible in κ:*

$$\Pr\left(\mathsf{Ve}_1(\mathsf{GC}, e) \neq \mathsf{Eval}(GC, \mathsf{En}(e,x)) \ : \ \begin{array}{c}(\mathsf{GC}, e, d) \leftarrow \mathcal{A}(1^\kappa, \mathcal{C}) \\ \mathsf{Ve}\,(\mathcal{C}, \mathsf{GC}, e) = \mathsf{accept}\end{array}\right)$$

Intuitively, the above definitions say that even a malicious constructor cannot create garbled circuits that are successfully verifiable (i.e., make Ve output accept) and at the same time violate the correctness condition. Moreover, one can extract the output wire key corresponding to output 1 given both input wire keys. This extraction requirement is natural in Yao's scheme which we define next. Looking ahead, in Yao's, knowing all input keys, one can iteratively "fully decrypt" every garbled gate and finally obtain the key corresponding to the output bit 1 even without computing an input x such that $\mathcal{C}(x) = 1$. We note that a natural and efficient way to obtain a verifiable garbling scheme is to generate GC by using the output of a pseudorandom generator on a seed as the random tape for Gb, and then provide the seed to the verification procedure Ve. Ve will regenerate the GC and the encoding and decoding tables, and will output accept for a garbled circuit if and only if it is equal to the generated one.

2.3.1 Yao's construction

A comprehensive treatment of Yao's construction of garbled circuits, was given in [LP09]. At a high-level, in Yao's construction, each wire of the boolean circuit is associated with two random strings called wire labels or wire keys that encode logical 0 and 1 wire values. A garbled truth table is constructed for every gate in the circuit, where each combination of input wire labels is used to encrypt the appropriate output wire label as per the gate functionality. This results in four ciphertexts per gate, one for each input combination of the gate. The evaluator knows only one label for each input wire, and can therefore, open only one of the four ciphertexts.

2.3.2 Free-XOR and other optimizations

Several works have studied optimizations to reduce the size of a garbled gate down from four ciphertexts. Garbled row-reduction was introduced by Naor, Pinkas and Sumner [NPS99]. There, instead of choosing the wire labels at random for each wire, they are chosen such that the first ciphertext will be the all-zero string, and hence need not be sent. In [PSSW09], the authors describe a way to further reduce the number of ciphertexts per gate to 2, by applying polynomial interpolation at each gate. Kolesnikov and Schneider [KS08a] introduced the Free XOR approach, allowing evaluation of XOR gates without any cost. Here, the idea is to choose wire labels such that the two labels on the same wire have the same (secret) offset across the entire circuit. The two labels for a given wire are of the form $(A, A \oplus \Delta)$, where Δ is secret and common to all wires. Now, as first proposed in [Kol05], an evaluator who has one of $(A, A \oplus \Delta)$ and one of $(B, B \oplus \Delta)$ can compute the XOR by simply XORing the wire labels. The result is either C

or $C \oplus \Delta$ where $C = A \oplus B$ and correctly represents the result of XOR. Thus, no ciphertexts are needed for the XOR gate. Kolesnikov, Mohassel and Rosulek proposed a generalization of Free XOR called FleXOR [KMR14]. In FleXOR, each XOR gate can be garbled using 0,1, or 2 ciphertexts, depending on certain structural properties of the circuit. In [ZRE15], the authors present a method that can garble an AND gate using only two ciphertexts. This technique is also compatible with Free XOR. The idea is to write an AND gate as a combination of XOR and two *half-gates*, where a half-gate is an AND gate for which one party knows one of the inputs. The half-gates can be garbled with one ciphertext each, and the resulting AND gate, in combination with free-XOR, uses two ciphertexts. In the above schemes, a hash/key-derivation function H is used for garbling, and different properties are required of H in the known garbling schemes. For example, the half-gates scheme requires either a circular-correlation-robust hash function, or works with a Davis-Meyer construction in the ideal cipher model.

2.4 Garbled Circuits for ZK

We review the approach of [JKO13] to construct zero-knowledge arguments for non-algebraic statements and some necessary building blocks.

2.4.1 Oblivious Transfer

Oblivious transfer (OT), first proposed by Rabin [Rab05] is a fundamental primitive for secure two-party and multi-party computation. It is a protocol between a sender and a receiver, where the sender has as inputs n secrets and the receiver holds a choice bit. In a 1-out-of-2 OT, the sender holds two inputs $s_0, s_1 \in \{0,1\}^k$ and the receiver holds a choice bit b. At the end of the protocol, the receiver obtains s_b. The sender learns nothing about the choice bit, and the receiver learns nothing about the sender's other input $s_{\tilde{b}}$.

Committing OT (COT). Several flavors of the OT primitive have been studied, like verifiable OT [Cré90], committed OT [CvT95], authenticated OT [NNOB12] and many others. In the variant we will use later, we need an OT protocol with a sender verifiability property- that is, at the end of the OTs, the sender is committed to its messages, and can be asked to reveal all its input messages to the receiver. This is closely related to the notion of committing OT [KS06], but can be achieved even more generally since we do not require individual commitments to sender's messages. In particular, as discussed in [JKO13] it can be satisfied by a protocol where the sender commits to a seed in the beginning of the protocol, and then runs any secure OT protocol using the output of a pseudorandom generator

on the seed as its random tape. Then the open phase can be realized by letting the sender reveal the seed and all the input messages. The ideal functionality $\mathcal{F}_{COT}$ is defined in Figure 2.1.

Figure 2.1: The ideal functionality $\mathcal{F}_{COT}$

- The receiver inputs $(choose, b), b \in \{0, 1\}$, and the sender inputs (m_0, m_1).
- Output m_b to the receiver.
- On input open from the sender, send (m_0, m_1) to the receiver.

2.4.2 ZK Proof Based on Garbled Circuits

Here, we review the garbled-circuit-based ZK protocol of Jawurek, Kerschbaum and Orlandi[JKO13]. To prove a statement $\exists w : R(x, w) = 1$ (for public R and x), the protocol proceeds as follows:

1. The verifier generates a garbled circuit computing $R(x, \cdot)$. Using a committing oblivious transfer, the prover obtains the wire labels corresponding to his private input w. Then the verifier sends the garbled circuit to the prover.
2. The prover evaluates the garbled circuit, obtaining a single garbled output (wire label). He commits to this garbled output.
3. The verifier opens his inputs to the committing oblivious transfer, giving the prover all garbled inputs. From this, the prover can check whether the garbled circuit was generated correctly. If so, the prover opens his commitment to the garbled output; if not, the prover aborts.
4. The verifier accepts the proof if the prover's commitment holds the output wire label corresponding to TRUE.

Security against a cheating prover follows from the properties of the circuit garbling scheme. Namely, the prover commits to the output wire label before the circuit is opened, so the *authenticity* property of the garbling scheme ensures that he cannot predict the TRUE output wire label unless he knows a w with $R(x, w) =$ TRUE. Security against a cheating verifier follows from correctness of the garbling scheme. The garbled output of a *correctly generated* garbled circuit reveals only the output of the (plain) circuit, and this garbled output is not revealed until the garbled circuit was shown to be correctly generated.

2.5 SNARKs for Arithmetic Circuits

A long line of work [Gro10, Lip12, BCCT12, GGPR13, BCCT13, PHGR13] has resulted in many proposals using pairing-based constructions that yield succinct non-interactive arguments where the argument itself consists of a constant number of group elements. They all rely on a common reference string and the so-called knowledge-extractor assumptions. In this section, we review some technical tools and assumptions that we rely on in later chapters.

2.5.1 Quadratic Arithmetic Programs

The work of [GGPR13] showed how to encode computations as quadratic programs. They show how to convert any Boolean circuit into a Quadratic Span Program (QSP), and any arithmetic circuit into a Quadratic Arithmetic Program (QAP). In this thesis, we will only use the latter definition.

Arithmetic circuits and QAPs. An arithmetic circuit consists of wires that carry values from a field $\mathbb{F}$, and are connected to addition and multiplication gates.

Definition 2.5.1 (Quadratic Arithmetic Program [GGPR13].)**.** *A quadratic arithmetic program (QAP) Q over a field $\mathbb{F}$ consists of three sets of polynomials $V = \{v_k(x) : k \in \{0, \cdots, m\}\}, W = \{w_k(x) : k \in \{0, \cdots, m\}\}, Y = \{y_k(x) : k \in \{0, \cdots, m\}\}$ and a target polynomial $t(x)$, all in $\mathbb{F}[X]$.*

Let $f : \mathbb{F}^n \to \mathbb{F}^{n'}$ be a function with input variables labeled $1, \cdots, n$ and output variables labeled $m - n' + 1, \cdots, m$. Q is said to compute f if the following holds: $a_1, \cdots, a_n, a_{m-n'+1}, \cdots, a_m \in \mathbb{F}^{n+n'}$ is a valid assignment to the input and output variables of f (i.e., $f(a_1, \cdots, a_n) = (a_{m-n'+1}, \cdots, a_m)$) iff there exist $(a_{n+1}, \cdots, a_{m-n'}) \in \mathbb{F}^{m-n-n'}$ such that, $t(x)$ divides $p(x)$ where,

$$p(x) = \left(v_0(x) + \sum_{k=1}^{m} a_k v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} a_k w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} a_k y_k(x)\right)$$

The size of the QAP Q is m, and degree is $deg(t(x))$.

The polynomials $v_k(x), w_k(x), y_k(x)$'s have degree at most $\deg(t(x)) - 1$, since they can be reduced modulo $t(x)$ without affecting the divisibility check.

Building a QAP. At a high level, we construct a QAP for a given arithmetic circuit C in the following way. For each multiplication gate g in C, we pick an arbitrary root $r_g \in \mathbb{F}$, and the target polynomial is defined to be $t(x) = \prod_g (x - r_g)$. The polynomials V, W, Y encode the left input, right input and output of each

gate respectively. If the kth wire is a left input to gate g, then $v_k(r_g) = 1$, and 0 otherwise. Similarly, $y_k(r_g) = 1$ if the kth wire is the output wire of gate g, and $y_k(r_g) = 0$ otherwise. For a gate g, we have,

$$\left(\sum_{k=1}^{m} a_k v_k(r_g)\right) \cdot \left(\sum_{k=1}^{m} a_k w_k(r_g)\right) = a_g y_k(r_g) = a_g$$

The above ensures that the value on the output wire of the gate is the product of its inputs which is the constraint for a multiplication gate. The divisibility check that $t(x)$ should divide $p(x)$ decomposes into $\deg(t(x))$ separate checks that $p(r_g) = 0$, one check for each gate g and root r_g of $t(x)$. The actual construction also handles addition and multiplication by constants. We refer the reader to [GGPR13, PHGR13] for a detailed discussion on building QAPs for arithmetic circuits. We point out that for any arithmetic circuit with d multiplication gates and N input/output elements, a QAP with degree d and size (number of polynomials in each set) $d + N$ can be constructed. Note that addition gates and multiplication-by-constant gates do not contribute to the size or degree of the QAP, and are thus, essentially for "free" in QAP-based SNARK constructions.

2.5.2 Bilinear Maps

Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_3$ be multiplicative cyclic groups of prime order p, and g_1, g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. A bilinear map is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$ with the following properties:

- Bilinearity: $\forall u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}, e(u^a, v^b) = e(u, v)^{ab}$.
- Non-degeneracy: $e(g_1, g_2) \neq 1$

One could set $\mathbb{G}_1 = \mathbb{G}_2$. However, we allow for the more general case where $\mathbb{G}_1 \neq \mathbb{G}_2$, in which case the map is called asymmetric bilinear map.

Let $\mathsf{GroupGen}$ be an asymmetric pairing group generator that on input 1^κ, outputs description of three cyclic groups $\mathbb{G}$, $\mathbb{H}$, $\mathbb{G}_T$ of prime order $p = \Theta(\kappa)$ equipped with a non-degenerate efficiently computable bilinear map $e : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$. It also outputs generators g and h for $\mathbb{G}$ and $\mathbb{H}$, respectively. We describe some q-type assumptions on bilinear maps below on which the security of zk-SNARKs relies.

Assumptions on Bilinear Maps.

Assumption 2.5.2 (q-PDH)**.** *The q-power Diffie-Hellman (q-PDH) assumption holds for* $\mathsf{GroupGen}$ *if for all non-uniform probabilistic polynomial time algorithms $\mathcal{A}$, the following probability is negligible in the security parameter.*

$$\Pr\left[g_1^{s^{q+1}} \leftarrow \mathcal{A}(\sigma) \; : \; \begin{array}{c} \sigma_1 = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{GroupGen}, \\ g_1 \leftarrow \mathbb{G}_1 \setminus \{1\}, g_2 \leftarrow \mathbb{G}_2 \setminus \{1\}, s \xleftarrow{R} \mathbb{Z}_p^*, \\ \sigma = (\sigma_1, g_1, g_2, g_1^s, g_2^s, g_1^{s^2}, g_2^{s^2}, \cdots, g_1^{s^q}, g_2^{s^q}, \\ g_1^{s^{q+2}}, g_2^{s^{q+2}} \cdots, g_1^{s^{2q}}, g_2^{s^{2q}}) \end{array} \right].$$

Assumption 2.5.3 (q-PKE)**.** *The q power-knowledge of exponent (q-PKE) assumption holds for* GroupGen *if for all non-uniform probabilistic polynomial time algorithms $\mathcal{A}$, there exists a non-uniform probabilistic polynomial time extractor $\chi_{\mathcal{A}}$ such that the following probability is negligible in the security parameter.*

$$\Pr\left[e(c^\alpha, g_2) = e(g_1, \hat{c}) \wedge c \neq \textstyle\prod_{i=0}^q g_1^{a_i s^i} \; : \; \begin{array}{c} (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{GroupGen}, \\ \sigma_1 = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e), \\ g_1 \leftarrow \mathbb{G}_1 \setminus \{1\}, g_2 \leftarrow \mathbb{G}_2 \setminus \{1\}, \\ \alpha, s \xleftarrow{R} \mathbb{Z}_p^*, \\ \sigma = (\sigma_1, g_1, g_2, g_1^s, g_2^s, \cdots, g_1^{s^q}, g_2^{s^q}, \\ g_1^{\alpha s}, g_2^{\alpha s}, \cdots, g_1^{\alpha s^q}, g_2^{\alpha s^q}), \\ (c, \hat{c}; a_0, \cdots, a_q) \leftarrow (\mathcal{A} || \chi_{\mathcal{A}}(\sigma, z)) \end{array} \right].$$

In the above, z is auxiliary information generated independently of α, and $(x; y) \leftarrow (\mathcal{A}||\chi_{\mathcal{A}})(\sigma, z)$ denotes that on input σ, $\mathcal{A}$ outputs x, and $\chi_{\mathcal{A}}$ given the same input σ, and $\mathcal{A}$'s random tape, outputs y.

Assumption 2.5.4 (q-SDH)**.** *The q-strong Diffie-Hellman (q-SDH) assumption holds for* GroupGen *if for all non-uniform probabilistic polynomial time algorithm $\mathcal{A}$, the following probability is negligible in the security parameter.*

$$\Pr\left[y = e(g_1, g_2)^{\frac{1}{s+c}}, c \in \mathbb{Z}_p^* \; : \; \begin{array}{c} (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{GroupGen}, \\ \sigma_1 = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e), \\ g_1 \leftarrow \mathbb{G}_1 \setminus \{1\}, g_2 \leftarrow \mathbb{G}_2 \setminus \{1\}, s \xleftarrow{R} \mathbb{Z}_p^*, \\ \sigma = (\sigma_1, g_1, g_2, g_1^s, g_2^s, g_1^{s^2}, g_2^{s^2}, \cdots, g_1^{s^q}, g_2^{s^q}), \\ y \leftarrow \mathcal{A}(\sigma) \end{array} \right].$$

2.5.3 zk-SNARK construction from QAP

We review the zk-SNARK construction of [PHGR13] known as Pinocchio, below. Let f be a function that maps N elements from $\mathbb{F}$ to 0 or 1. Convert f into an arithmetic circuit C and build a QAP $Q = (V, W, Y, t(x))$ for C of size m and degree d. We let the indices $i \in [1, n]$ denote the public input (the statement y) and $i \in [n+1, N]$ denote the private input (the witness x).

1. **CRS generation**. Choose $r_v, r_w, \alpha_v, \alpha_w, \alpha_y, s, \beta, \gamma \xleftarrow{R} \mathbb{F}$. Set $r_y = r_v r_w, g_v = g^{r_v}, g_w = g^{r_w}$, and $g_y = g^{r_y}$. Set the CRS to be:

$$\begin{aligned}\mathsf{crs} = \Big(&\{g_v^{v_k(s)}\}_{k\in[n+1,m]}, \{g_w^{w_k(s)}\}_{k\in[n+1,m]}, \{g_y^{y_k(s)}\}_{k\in[n+1,m]},\\ &\{g_v^{\alpha_v v_k(s)}\}_{k\in[n+1,m]}, \{g_w^{\alpha_w w_k(s)}\}_{k\in[n+1,m]}, \{g_y^{\alpha_y y_k(s)}\}_{k\in[n+1,m]},\\ &\{g^{s^i}\}_{i\in[d]}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k\in[n+1,m]}\Big).\end{aligned}$$

   Set the short verification CRS to be:

$$\begin{aligned}\mathsf{shortcrs} = \big(&g, g^{\alpha_v}, g^{\alpha_w}, g^{\alpha_y}, g^{\gamma}, g^{\beta\gamma}, g_y^{t(s)}, \{g_v^{v_k(s)}\}_{k\in\{0\}\cup[n]},\\ &\{g_w^{w_k(s)}\}_{k\in\{0\}\cup[n]}, \{g_y^{y_k(s)}\}_{k\in\{0\}\cup[n]}\big).\end{aligned}$$

2. **Prove**. On input statement y, witness x, and crs, the prover evaluates the QAP to obtain $\{a_i\}_{i\in[m]}$. (Equivalently, evaluates C to obtain the values on the circuit wires). The prover solves for the quotient polynomial h such that $p(x) = h(x)t(x)$. Let $v_{mid}(x) = \sum_{k\in[n+1,m]} a_k v_k(x)$, and similarly define $w_{mid}(x)$ and $y_{mid}(x)$. The prover computes the proof π:

$$\begin{aligned}\big(&g_v^{v_{mid}(s)}, g_w^{w_{mid}(s)}, g_y^{y_{mid}(s)}, g^{h(s)},\\ &g_v^{\alpha_v v_{mid}(s)}, g_w^{\alpha_w w_{mid}(s)}, g_y^{\alpha_y y_{mid}(s)},\\ &g_v^{\beta v_{mid}(s)} g_w^{\beta w_{mid}(s)} g_y^{\beta y_{mid}(s)}\big)\end{aligned}$$

3. **Verify**. On input $\mathsf{shortcrs}$, y, and a proof $\pi = (g_v^{V_{mid}}, g_w^{W_{mid}}, g_y^{Y_{mid}}, g^H, g_v^{V'_{mid}}, g_w^{W'_{mid}}, g_y^{Y'_{mid}}, g^Z)$:

   - Compute $g_v^{v_{in}(s)} = \prod_{k\in[n]} \left(g_v^{v_k(s)}\right)^{a_k}$. Similarly, compute $g_w^{w_{in}(s)}$ and $g_y^{y_{in}(s)}$. Check whether,

$$\begin{aligned}&e(g_v^{v_0(s)} g_v^{v_{in}(s)} g_v^{V_{mid}}, g_w^{w_0(s)} g_w^{w_{in}(s)} g_w^{W_{mid}})\\ &= e(g_y^{t(s)}, g^H) \cdot e(g_y^{y_0(s)} g_y^{y_{in}(s)} g_y^{Y_{mid}}, g).\end{aligned}$$

   - Verify that $e(g_v^{V'_{mid}}, g) = e(g_v^{V_{mid}}, g^{\alpha_v})$, $e(g_w^{W'_{mid}}, g) = e(g_w^{W_{mid}}, g^{\alpha_w})$, and $e(g_y^{Y'_{mid}}, g) = e(g_y^{Y_{mid}}, g^{\alpha_y})$.
   - Verify $e(g^Z, g^\gamma) = e(g_v^{V_{mid}} g_w^{W_{mid}} g_y^{Y_{mid}}, g^{\beta\gamma})$.

   Output 1 if all the verifications succeed, else output 0.

Chapter 3

ZK for Combination Statements[1]

In this chapter, we give protocols to use a garbled circuit or SNARK to prove a statement is true, and prove that the input to the statement is the committed value in an algebraic commitment. In Section 3.1, we show how to use garbled circuit for the Boolean circuit component of the statement, and sigma protocol for the algebraic components. In Section 3.2, we give protocols to use a SNARK for the circuit component of the statement and prove consistency of the input to SNARK with a committed value.

3.1 Sigma Protocols and GC for Combination Statements

3.1.1 Preliminaries

Simulation-based Security. We use a simulation-based definition of security in the ideal/real world paradigm, which is formulated by specifying an ideal functionality. A protocol is secure if it "emulates" this ideal functionality in the presence of any adversary. Our definitions are in the stand-alone setting (as opposed to the UC framework). We formulate the simulation-based definitions by defining a functionality $\mathcal{F}$ in the ideal world. In the ideal world, all parties and the adversary $\mathcal{A}$ interact via $\mathcal{F}$. Let $IDEAL_{\mathcal{F},\mathcal{A}}(x_1, x_2)$ denote the output vector of the adversary and the honest party from the execution in the ideal world. In the real world, a protocol π is executed among the parties, and let $REAL_{\pi,\mathcal{A}}(x_1, x_2)$ denote the output of the adversary and the honest party from the execution of π. A two party protocol π securely realizes the functionality $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ in the

[1] This chapter is based on joint work with Melissa Chase and Payman Mohassel that appeared in CRYPTO 2016 [CGM16], and joint work with Shashank Agrawal and Payman Mohassel that is yet to be published [AGM17]. Some passages are taken verbatim from these sources.

real world, there exists a PPT adversary $\mathcal{S}$ in the ideal-world, such that

$$\{IDEAL_{\mathcal{F},\mathcal{S}}(x_1, x_2)\}_{x_1,x_2 s.t |x_1|=|x_2|} \stackrel{c}{\equiv} \{REAL_{\pi,\mathcal{A}}(x_1, x_2)\}_{x_1,x_2 s.t |x_1|=|x_2|}$$

that is, the two distributions are computationally indistinguishable.

Commitment Scheme. A commitment protocol involves two parties: the committer and the receiver. At a high level, it consists of two stages, a commitment phase and a de-commitment phase. In the commitment stage, the committer with a secret input m engages in a protocol with the receiver. At the end of this protocol, receiver does not know what m is (hiding property), and at the same time, the committer, can subsequently in the de-commitment phase, open only one possible value of m (binding property). Throughout, we use algebraic commitment schemes that allow proving linear relationships among committed values. An example of such a scheme with computational binding and unconditional hiding properties based on the discrete logarithm problem is the one due to Pedersen [Ped91]. It works in a group G of prime order q. Given two random generators g and h such that $\log_g h$ is unknown, a value $x \in \mathbb{Z}_q$ is committed to by choosing r randomly from $\mathbb{Z}_q$, and computing $C_x = g^x h^r$. Protocols are known in literature to prove knowledge of a committed value, equality of two committed values, and so on, and the protocols can be combined in natural ways. In particular, Pedersen commitments allows proving linear relationships among committed values: Given C_x and C_y, prove that $y = ax + b$ for some public values a and b.

3.1.2 Proving Non-algebraic Statements on Algebraic Commitments

Garbled circuits are secure against semi-honest adversaries, and can be made secure against malicious adversaries by using generic techniques like the cut-and-choose technique. Here, the garbled circuit constructor sends multiple copies of the circuit to the circuit evaluator, who chooses a random subset of the received circuits. Then the circuit constructor "opens" the chosen garbled circuits and now the evaluator may verify that they indeed garble the correct function, and then the remaining unopened garbled circuits are evaluated. The complexity of this protocol grows with the security parameter and is inefficient in practice.

In [JKO13], the authors exploit the fact that the verifier's inputs need not be kept secret. Therefore, the verifier can open the circuit and prove that a circuit that computes the correct function was garbled, thereby eliminating the need for more check circuits as in classic cut-and-choose for security against malicious adversaries. In our application, the verifier does have a private input, but it need not be kept secret *post evaluation*. Our constructions also use the technique of [JKO13] to

open the evaluated circuit, and achieve malicious security with only one garbled circuit.

An important sub-protocol used in our constructions, is to commit to an input x using an algebraic commitment $\mathsf{Com}(x)$ (e.g. Pedersen commitment), and perform a zero-knowledge proof of a non-algebraic statement about x, i.e. that $f(x) = 1$ for a boolean circuit f.

Such a protocol allows one to efficiently switch between proving algebraic statements on a committed input (e.g. proof of knowledge of a signature on a committed input) and non-algebraic statement (e.g. hashing, comparison, equality testing and more).

The protocols in this section are defined in terms of an ideal functionality, and are proven secure in the ideal/real world paradigm. We start by defining the above task in terms of an ideal functionality in Figure 3.1. We provide two instantiations for this functionality that provide different efficiency trade-offs depending on the input size and the algebraic commitment scheme used.

Figure 3.1: The ideal functionality $\mathcal{F}_{\mathsf{Com},f}$

- The verifier inputs $\mathsf{Com}(x)$ and prover inputs the opening information x and the randomness for $\mathsf{Com}(x)$.
- If $f(x) = 1$ and the opening to the commitment verifies, output accept to the verifier.

Note that in the protocol of [JKO13] described in Section 2.4.2, the prover evaluates the garbled circuit on an input which is completely known to him. This is the main reason that the garbled circuit used for evaluation can also be later opened and checked for correctness, unlike in the setting of cut-and-choose for general 2PC. Along the same lines, it was further pointed out in [FNO15] that the circuit garbling scheme need not satisfy the *privacy* requirement of [BHR12], only the *authenticity* requirement. Removing the privacy requirement from the garbling scheme leads to a non-trivial reduction in garbled circuit size.

In one of our constructions (Section 3.1.4), the verifier does have a private input, but its input only needs to be kept private until the circuit is evaluated and the prover has committed to the output. In that scenario, we also invoke the *privacy* property of the garbling scheme. The state of the art garbling scheme uses the free-XOR technique [KS08b] to garble XOR gates and the half-gate technique to garble AND gates [ZRE15]. For a circuit with g non-XOR gates, the total number of ciphertexts is $2g$, and the number of hash invocations is $4g$ for the garbler and $2g$ for the evaluator.

For *privacy-free* garbling, the costs are reduced by factor of two (see [FNO15,

ZRE15]). In particular, for a circuit with g non-XOR gates, the total number of ciphertexts is g, and the number of hash invocations is $2g$ for the garbler and g for the evaluator.

We need to garble a few common building-block circuits in our constructions. It is helpful to review the size of these circuits based on the concrete constructions given in [KSS09]. The circuit for comparing ℓ bit integers requires 4ℓ non-XOR gates. The circuit for testing equality of ℓ-bit integers also requires 4ℓ non-XOR gates. The circuit for adding two ℓ-bit integers requires 4ℓ non-XOR gates, while the circuit for multiplying two ℓ-bit integers requires $8\ell^2 - 4\ell$ non-XOR gates.

The starting point for both instantiations is the ZK-proof of non-algebraic statements based on garbled circuits [JKO13] (see Section 2.4.2). As a naive solution we could garble a circuit that takes x and the opening of $\mathsf{Com}(x)$ as prover's input and outputs 1 if $f(x) = 1$ and $\mathsf{Com}(x)$ correctly opens to x. The main drawback of this solution is that checking correctness of opening for an algebraic commitment requires performing expensive group operations (e.g. exponentiation) inside the garbled circuit which would dominate the computation/communication cost. We provide two instantiations of $\mathcal{F}_{\mathsf{Com},f}$ in Sections 3.1.3 and 3.1.4 that avoid these costs and perform all algebraic operations outside the garbled circuit. In Section 3.1.5, we discuss the efficiency considerations of the two protocols.

3.1.3 First Protocol

In our first construction, we have the prover commit to each bit of x, i.e. $\mathsf{Com}(x_i)$ for all $i \in [n]$, and prove that when combined, they yield x.

Then, following the GC-based approach, the verifier constructs a garbled circuit that computes $f(x)$, parties go through the steps of the GC-based ZK proof for the prover to prove knowledge of a value x' such that $f(x') = 1$. The main issue is that a malicious prover may use a different input $x' \neq x$ in the circuit than what he committed to.

But we observe that the input keys associated with x' in the GC (which are obtained through the COT), can function as one-time MACs on each bit of x' and can be used to enforce that $x' = x$ using efficient algebraic ZK proofs that take place outside the garbled circuit. In particular, immediately after the COTs, the prover commits to its input keys i.e. $X_i^{x'_i}$ for the ith bit of x'. When the GC is opened and both input keys X_i^0, X_i^1 are opened, the prover can provide ZK proofs that $X_i^{x'_i} = x_i X_i^1 + (1 - x_i) X_i^0$ if the commitment scheme provides efficient proofs of linear relations.

A small subtlety is that a malicious prover may use the integer $x' = x + q$ as input to the circuit where q is the group size for the commitment scheme, but commit to x in the commitment outside of the garbled circuit. A simple way of preventing this is to allow exactly $\lfloor \log q \rfloor$ input wires for x in the circuit. This

would guarantee that $x < q$ for free but is not ideal as it does not accept any value between $2^{\lfloor \log q \rfloor}$ and q. Alternatively, one can augment the garbled circuit with a comparison circuit that outputs $x < q$. For simplicity we assume the first variant in both instantiations (Figure 3.2 and Figure 3.3), but discuss the cost of such a comparison circuit in Section 3.1.5.

The complete protocol description in the COT-hybrid model is given in Figure 3.2. We point out that steps 1, 6 and 14 are additions compared to the protocol of [JKO13].

Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Eval}, \mathsf{Ve})$ be a verifiable garbling scheme. Let F be the following functionality: it takes as input x, and outputs v such that $v = 1$ if $f(x) = 1$ and 0 otherwise. The verifier is in possession of $C_x = \mathsf{Com}(x)$, the prover has input x, the randomness to open C_x, and both parties have as input the security parameter κ.

1. The prover commits to the bits of x by sending bit-wise commitment to x: $C_i = \mathsf{Com}(x_i), \forall 1 \leq i \leq n$.
2. The verifier constructs a garbled circuit for F.
$$(GC, e, d) \leftarrow \mathsf{Gb}(1^\kappa, F)$$
3. The verifier inputs the wire labels corresponding to the prover's input by sending (i, X_i^0, X_i^1) for all $i \in [n]$ to $\mathcal{F}_{COT}$.
4. The prover inputs his choice bits by sending (i, x_i) for all $i \in [n]$ to $\mathcal{F}_{COT}$.
5. $\mathcal{F}_{COT}$ outputs X_i' for all $i \in [n]$ to the prover where $X_i' = X_i^{x_i}$.
6. The prover commits to the received input wire labels by sending $C_{X_i} = \mathsf{Com}(X_i')$ for all i.
7. The verifier sends the garbled circuit GC to the prover.
8. The prover evaluates the garbled circuit
$$Z \leftarrow \mathsf{Eval}(GC, \{X_i'\}_{i \in [n]})$$
9. The prover commits to the garbled output Z by sending $\mathsf{Com}(Z)$ to the verifier and proves knowledge of opening.
10. The verifier sends open to $\mathcal{F}_{COT}$.
11. $\mathcal{F}_{COT}$ sends (X_i^0, X_i^1) to the prover for all $i \in [n]$.

12. The prover verifies that the correct circuit was garbled by running $\mathsf{Ve}(F, GC, \{X_i^0, X_i^1\}_{i\in[n]})$. If the output is not accept, the prover terminates. Otherwise if Ve outputs accept, he opens the commitment to the output Z by sending Z and the randomness used in $\mathsf{Com}(Z)$.

13. The verifier checks that the opening is correct and that $\mathsf{De}(d, Z) = 1$. If the opening is not correct or if $\mathsf{De}(d, Z) \neq 1$, the verifier outputs reject and terminates.

14. If the verifier did not terminate, the prover and the verifier engage in a Zero-knowledge protocol to prove the following:

    - $\mathsf{PK}\{(x_i, X_i', r, R) : C_i = \mathsf{Com}(x_i) \wedge C_{X_i} = \mathsf{Com}(X_i') \wedge X_i' = x_i X_i^1 + (1 - x_i)X_i^0\}$, $\forall 1 \leq i \leq n$.
    - $\mathsf{PK}\{(x, x_1, \cdots, x_n, r, r_1, \cdots r_n) : C_x = \mathsf{Com}(x) \wedge C_i = \mathsf{Com}(x_i) \wedge x = \sum 2^i x_i\}$

15. If the zero-knowledge proof verifies, the verifier outputs accept.

Figure 3.2: The Protocol $\Pi_{\mathsf{Com},f}$

Theorem 3.1.1. *Let $\mathcal{G}$ be a garbling scheme satisfying correctness and authenticity properties as defined in Section 2.3. Let Com be a secure commitment scheme, and let the proofs PK be implemented with a zero knowledge proof of knowledge. Then, the protocol $\Pi_{\mathsf{Com},f}$ in Figure 3.2 securely implements $\mathcal{F}_{\mathsf{Com},f}$ in the presence of malicious adversaries in the $\mathcal{F}_{COT}$-hybrid model.*

Proof. **Corrupt Prover P^*.**

The simulator works as follows: It extracts the prover's input x' sent to the $\mathcal{F}_{COT}$ functionality in step 4. It then plays the role of the honest verifier in the rest of the simulation - it constructs the garbled circuit honestly and uses its input keys as verifier's inputs to the COT functionality. The simulator then extracts a value Z committed to by the prover from the proofs of knowledge of opening in step 9. It also extracts prover's committed input x and the values X_i' that prover committed to in the protocol, using the extractor for the ZK proof of knowledge in step 14. The simulator finally outputs as witness x and the opening extracted from the ZK proofs of step 14, iff all the following hold: $x = x'$, $f(x) = 1, Z$ is the one-key of the output wire, $X_i' = X_i^{x_i}$ for all i, the commitment in step 9 is opened to Z, and the ZK proofs of step 14 verifies. Note that in the ideal model, the functionality will always output accept when the simulator sends this witness.

We now prove that P^*'s view in the real protocol is indistinguishable from his view with the simulator via a series of intermediate games.

- Game Ideal: This is the interaction P^* with the simulator and functionality as described above.

- Game $\mathcal{G}_0$: This is the interaction of P^* with the simulator as described above, with the exception that instead of the simulator sending x and the opening to the functionality which outputs accept iff $f(x) = 1$, the game will output accept iff $f(x') = 1$ for the x' extracted from the OT (and all the other conditions listed hold). Since one of the conditions checks $x = x'$, this is identical.

- Game $\mathcal{G}_1$: This game behaves exactly as in $\mathcal{G}_0$ except for a slight change in the accept condition. It outputs accept if $f(x') = 1$ and $X'_i = X_i^{x_i}$ for all i and Z is the one-key of the output wire and the commitment in step 9 is correctly opened to Z, and all the ZK proofs verify (i.e. no $x = x'$ check).

  Indistinguishability:

  Define the event Bad as the event that $x \neq x', f(x') = 1, Z$ is the one-key of the output wire, $K'_i = K_i^{x_i}$ for all i, and the opening to Z is correct and the ZK proofs of step 14 verify.

  Observe that $\mathcal{G}_0$ is identical to $\mathcal{G}_1$ conditioned on $\overline{\mathsf{Bad}}$. We now argue that $\Pr[\mathsf{Bad}]$ is negligible, by observing that an adversary who makes us reject $\mathcal{G}_0$ but accept in $\mathcal{G}_1$, can only succeed with probability $1/2^s$ where s is a statistical security parameter, in the COT hybrid model. Without loss of generality, let us assume the ith bit of x is 0 and ith bit of x' is 1. Then, the probability of the adversary guessing X_i^0 given only X_i^1 is less than $1/2^{|K_i^0|}$. Note that $|X_i^0|$ is the computational security parameter.

  Hence Games $\mathcal{G}_0$ and $\mathcal{G}_1$ are indistinguishable except with negligible probability in s.

- Game $\mathcal{G}_2$: This game behaves as in $\mathcal{G}_1$ except for another change in the accept condition. We accept if $f(x') = 1$ and ZK proofs of step 14 verifies and Z is the one-key of the output wire, and the commitment in step 9 is correctly opened to Z (i.e. no $X'_i = X_i^{x_i}$ check).

  If an adversary can distinguish between Games $\mathcal{G}_1$ and $\mathcal{G}_2$, we can break the *soundness of the ZK proof* of step 14. Therefore, $\mathcal{G}_1$ and $\mathcal{G}_2$ are indistinguishable.

- Game $\mathcal{G}_3$: This game behaves as in $\mathcal{G}_2$ except for a small change in accept condition. We accept if ZK proofs of step 14 verifies and Z is the one-key of the output wire, and the commitment in step 9 is correctly opened to Z (i.e. no $f(x') = 1$ check).

Games $\mathcal{G}_2$ and $\mathcal{G}_3$ are identical, except when the following event occurs: $f(x') \neq 1$ and ZK proof of step 14 passes, and Z is the one-key of the output wire. When this event occurs, we accept in $\mathcal{G}_3$ and reject in $\mathcal{G}_2$. We now argue that the probability of this event is negligible. For the sake of contradiction, assume the prover's input to OT is x' such that $f(x') \neq 1$, but the value committed to is the correct one-key Z for the output wire. We can use such a prover to break the authenticity of the garbling scheme (See Definition 2.3.3).

- Game $\mathcal{G}_4$: This game behaves as in $\mathcal{G}_3$ except for the accept condition. We accept if the ZK proofs of step 14 verifies and the commitment in step 9 opens correctly (i.e. no check that it is the same as extracted Z).

  An adversary who can distinguish between $\mathcal{G}_3$ and $\mathcal{G}_4$ can be used to violate the binding property of the commitment scheme.

  $\mathcal{G}_4$ is identical to the real world game with an honest verifier.

Corrupt Verifier V^*. The simulator plays the role of the prover and commits to bits of a random value in step 1. It also uses a random value as prover's inputs to the COT, and receives the verifier's inputs to the COT functionality (X_i^0, X_i^1) for all i, i.e. the input keys to the GC. The simulator then commits to the keys corresponding to the random input it used in the OTs.

It runs $\mathsf{Ve}(GC, (X_i^0, X_i^1), f)$ to either obtain reject, or accept. If the output is reject it commits to a dummy value, else it runs the extractor $\mathsf{Ve}_1(GC, (X_i^0, X_i^1), f)$ to obtain the one-key for the output wire denoted by Z, and commits to Z.

It then receives the "open" message from the verifier. If Ve had not output reject earlier, the simulator opens the commitment to Z and uses the simulator for the ZK proof to simulate the proofs of step 14. Otherwise, the simulator aborts.

- Game $\mathcal{G}_0$: This is the interaction of V^* with the simulator as described above.

- Game $\mathcal{G}_1$: Is similar to game $\mathcal{G}_0$ except that the real input x of the prover is committed to.

  The two games are indistinguishable due to the hiding property of the commitment scheme.

- Game $\mathcal{G}_2$: Is similar to $\mathcal{G}_1$ except that instead of computing Z by running Ve, we run $\mathsf{Eval}(GC, X_i^{x_i})$ to compute and commit to Z.

  The two games are indistinguishable due to the second condition in the *correctness* property of the garbling scheme. Note that we are also implicitly using the committing OT property (the protocol described in the COT hybrid model) as the keys extracted in the OTs and what the functionality sends to the honest prover are the same.

- Game $\mathcal{G}_3$: Is similar to $\mathcal{G}_2$ except that the honest input x of the prover is used in the OTs.

  The two games are identical in the OT hybrid model.

- Game $\mathcal{G}_4$: Is similar to $\mathcal{G}_3$ except that the simulator commits to the input keys associated with the real input x.

  The two games are identical due to the hiding property of the commitment scheme.

- Game $\mathcal{G}_5$: Is similar to $\mathcal{G}_4$ except that in step 14, the simulator performs the proofs honestly.

  The two games are indistinguishable due to the zero-knowledge property of the ZK proof.

  Note that $\mathcal{G}_5$ is the real game with the honest prover.

□

3.1.4 Second Protocol

We now give an alternative construction that implements the functionality in Figure 3.1. In particular, we avoid the bit-wise commitments to each bit of x_i, and the associated bit-wise ZK proofs, and hence require fewer public-key operations (exponentiations) in the construction. On the other hand, the garbled circuit is augmented and hence a larger number of symmetric-key operations are needed.

The idea is as follows. In order to ensure that the prover uses the same input x in the GC, we have the circuit not only compute $f(x)$ but also a one-time MAC of x, i.e. $t = ax + b$ for random a and b of the verifier's choice. The values a and b are initially unknown to the prover, but are opened along with the GC after the prover has committed to t. Given a and b, the prover then provides a ZK proof that $\mathsf{Com}(t)$ is indeed the one-time MAC of x (using efficient proofs of linear relations). We note that the $t = ax + b$ operation performed in the circuit is on integers.

We note that our construction deviates from the standard construction of GC-based ZK where the verifier has no input to the garbled circuit, and privacy-free garbling is sufficient. In particular, we do invoke the privacy property of the garbling scheme in our construction to ensure that the prover does not learn a and b until the opening stage.

The complete protocol description in the COT-hybrid model is given in Figure 3.3.

Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Eval}, \mathsf{Ve})$ be a garbling scheme. Let F be the following functionality: it takes as inputs x, a, b and outputs v, t such that $v = 1$ if $f(x) = 1$ and 0 otherwise, and $t = ax + b$. The verifier is in possession of $C_x = \mathsf{Com}(x)$, the prover has input x and the randomness to open C_x. Both parties have as input the security parameter κ.

1. The verifier generates uniformly random integers a and b of length s and $n+s$ respectively, where $n = |x|$. It commits to them by sending $C_a = \mathsf{Com}(a)$, $C_b = \mathsf{Com}(b)$ and proves knowledge of their opening.

2. The verifier constructs a garbled circuit for F.

$$(GC, e, d) \leftarrow \mathsf{Gb}(1^\kappa, F(x, a, b) = (f(x), ax + b))$$

3. The prover inputs his choice bits by sending (i, x_i) for all $i \in [n]$ to $\mathcal{F}_{COT}$.

4. The verifier inputs the wire keys corresponding to the prover's input by sending (i, X_i^0, X_i^1) for all $i \in [n]$ to $\mathcal{F}_{COT}$.

5. $\mathcal{F}_{COT}$ outputs X_i' for all $i \in [n]$ to the prover where $X_i' = X_i^{x_i}$.

6. The verifier sends the garbled circuit GC to the prover. Note that in what follows, for simplicity, we consider the input keys for a and b to be part of the GC itself, and hence not sent separately.

7. The prover evaluates the garbled circuit

$$(t', Z) \leftarrow \mathsf{Eval}(GC, \{X_i'\}_{i \in [n]})$$

8. The prover commits to the garbled output Z by sending $\mathsf{Com}(Z)$ to the verifier and proves knowledge of opening.

9. The verifier sends the decoding information d_t for t.

10. The prover decodes

$$t = \mathsf{De}(d_t, t')$$

and commits to the decoded output by sending $C_t = \mathsf{Com}(t)$, and proves knowledge of opening.

11. The verifier sends open to $\mathcal{F}_{COT}$.

12. $\mathcal{F}_{COT}$ sends (X_i^0, X_i^1) to the prover for all $i \in [n]$.

13. The verifier opens $\mathsf{Com}(a)$ and $\mathsf{Com}(b)$. The prover checks the openings and aborts if they fail.

14. The prover verifies that the correct circuit was garbled by running $\mathsf{Ve}(GC, \{X_i^0, X_i^1\}_{i\in[n]}, F)$. It also checks that garbled inputs for x, a, b are the correct one. If any of checks fail, the prover terminates. Otherwise, it receives the decoding vector d, and he opens the commitment to the output Z by sending Z and randomness.

15. The verifier checks that the opening is correct and that $\mathsf{De}(d, Z) = 1$. If the opening is not correct or if $\mathsf{De}(d, Z) \neq 1$, the verifier outputs reject and terminates.

16. If the verifier did not terminate, the prover and the verifier engage in a Zero-knowledge protocol to prove the following:

$$\mathsf{PK}\{(x, t, r, R) : C_x = \mathsf{Com}(x) \wedge C_t = \mathsf{Com}(t) \wedge t = ax + b\}$$

17. If the zero-knowledge proof verifies, the verifier outputs accept.

Figure 3.3: The Protocol $\Pi_{\mathsf{MAC},f}$

Theorem 3.1.2. *Let $\mathcal{G}$ be a garbling scheme satisfying correctness, authenticity, and privacy properties as defined in Section 2.3. Let* Com *be a secure commitment scheme, and let the proofs PK be implemented with a zero knowledge proof of knowledge. Then, the protocol $\Pi_{\mathsf{MAC},f}$ in Figure 3.3 securely implements $\mathcal{F}_{\mathsf{Com},f}$ in the presence of malicious adversaries in the $\mathcal{F}_{COT}$-hybrid model.*

Proof. **Corrupt Prover P^*.**

The simulator works as follows: It uses the OT simulator to extract the prover's input x' to the OT. It then plays the role of the honest verifier in the rest of the simulation - it chooses a, b randomly as the honest verifier would, constructs the garbled circuit honestly and uses its input keys as verifier's inputs to the COT functionality. The simulator then extracts the value Z' committed to by the prover from the proofs of knowledge of opening in step 8. It also extracts prover's committed input x and the tag t' that the prover committed to in the protocol, using the extractor for the ZK proof of knowledge in step 16. The simulator finally outputs x and the opening extracted from the ZK proofs, iff all the following hold: $x = x'$, $f(x) = 1$, Z is the one-key of the output wire, $t' = ax + b$, the commitment in step 8 is opened to Z, and the ZK proof of step 16 verifies. Note that in the ideal model the functionality will always output accept when the simulator sends this witness.

We now prove that a corrupt prover's view in the real protocol is indistinguishable from his view with the simulator by a series of intermediate games.

- Game Ideal: This is the interaction of the corrupt prover with the simulator and functionality as described above.

- Game $\mathcal{G}_0$: This is the interaction of the corrupt prover with the simulator as described above, with the exception that instead of the simulator sending x and the opening to F, which outputs accept iff $f(x) = 1$, the game will output accept iff $f(x') = 1$ for the x' extracted from the OT (and all the other conditions listed hold). Since one of the conditions checks $x = x'$, this is identical.

- Game $\mathcal{G}_1$: In this game, the simulator behaves exactly as in G_0 except that it does not check the $x = x'$ condition.

  Define the event Bad as the event that $x \neq x'$ but $t = ax + b$. Observe that $\mathcal{G}_0$ is identical to $\mathcal{G}_1$ conditioned on $\overline{\mathsf{Bad}}$. We argue that Pr[Bad] is negligible due to the unforgeability property of the one-time MAC, the hiding property of the commitment scheme, and the privacy of the garbled circuit.

  Consider a game where we run as in $\mathcal{G}_1$ but stop after step 10, and look at the probability that in this game $t' = ax + b$ but $x \neq x'$; if Pr[Bad] is non-negligible, this will be non-negligible as well. Now, by the privacy of the garbled circuit, this is indistinguishable from a game where the verifier computes a tag t on x', and then constructs (GC, e, d) using the privacy simulator: $\mathcal{S}(F, (t, 1))$. Similarly, by the hiding of the commitment scheme this is still indistinguishable from a game where the verifier commits to random values instead of a, b. But if in this final game we get $t' = ax + b$ and $x \neq x'$ with non-negligible probability, then we can break the unforgeability of the MAC. The probability of forgery is bounded by $1/2^{|a|}$, and hence exponentially small in the statistical security parameter $s = |a|$.

- Game $\mathcal{G}_2$: In this game, the simulator behaves as in $\mathcal{G}_1$ except that it does not check the condition $t = ax + b$.

  If an adversary can distinguish between Games $\mathcal{G}_2$ and $\mathcal{G}_1$, we can break the soundness of the ZK proof of step 16.

- Game $\mathcal{G}_3$: In this game, the simulator behaves as in $\mathcal{G}_2$ except that we do not check the condition $f(x') = 1$.

  Games $\mathcal{G}_2$ and $\mathcal{G}_3$ are identical, except when the following event occurs: $f(x') \neq 1$ and ZK proof of tag verifies and Z is the one-key of the output wire. We now argue that the probability of this event is negligible. For the sake of contradiction, assume the prover's input to OT is x' such that $f(x') \neq 1$, but the value committed to is the correct one-key Z for the output wire. We can use such a prover to break the authenticity of the garbling scheme (See definition 2.3.3).

- Game $\mathcal{G}_4$: In this game, the simulator behaves as in $\mathcal{G}_3$ except for the accept condition. The simulator accepts if the ZK proofs of step 16 verifies and the commitment in step 8 opens correctly (i.e. no check that it is the same as extracted Z).

  An adversary who can distinguish between $\mathcal{G}_4$ and $\mathcal{G}_3$ can be used to violate the binding property of the commitment scheme.

  $\mathcal{G}_4$ is identical to the real world game with an honest verifier.

Corrupt Verifier V^*. The simulator extracts a and b from the proofs of knowledge of their openings by verifier. It uses a random value as prover's inputs to the COT, and receives the verifier's inputs to the COT functionality (X_i^0, X_i^1) for all i, i.e. the input keys to the verifier GC.

It then runs $\mathsf{Ve}(GC, (X_i^0, X_i^1), F)$ to either obtain reject or accept. If the GC verifies, the simulator runs $\mathsf{Ve}_1(GC, (X_i^0, X_i^1), F)$ (and checks against the extracted a, b) to extract the decoding information d. If the output is reject it commits to dummy values for Z and t, else it commits to the one-key for the output wire denoted by Z, and a dummy t.

The simulator receives the openings of $\mathsf{Com}(a)$ and $\mathsf{Com}(b)$. If the openings are not what it extracted earlier, or if Ve had output reject earlier, it aborts. Else, the simulator opens the commitment to Z and uses the simulator for the ZK proof to simulate the proofs of step 16.

- Game $\mathcal{G}_0$: This is the interaction of the corrupt verifier with the simulator as described above.

- Game $\mathcal{G}_1$: Is similar to game $\mathcal{G}_0$ except that $t = ax + b$ for the real input x of prover is committed to.

  The two games are indistinguishable due to the hiding property of the commitment scheme.

- Game $\mathcal{G}_2$: Is similar to $\mathcal{G}_1$ except that instead of computing Z and t by running Ve, we run $\mathsf{Eval}(GC, X_i^{x_i})$ to compute and commit to Z and t.

  The two games are indistinguishable due to the second condition in the *correctness* property of the garbling scheme, and binding property of commitments $\mathsf{Com}(a)$ and $\mathsf{Com}(b)$. Note that we are also implicitly using the committing OT property (the protocol described in the COT hybrid model) as the keys extracted in the OTs and what the functionality sends to the honest prover are the same.

- Game $\mathcal{G}_3$: Is similar to $\mathcal{G}_2$ except that the honest input x of the prover is used in the OTs.

  The two games are identical in the OT hybrid model.

- Game $\mathcal{G}_4$: Is similar to $\mathcal{G}_3$ except that in step 14, the simulator performs the proofs honestly.

  The two games are indistinguishable due to zero-knowledge property of the ZK proof.

  Note that $\mathcal{G}_4$ is the real game with the honest prover.

□

3.1.5 Efficiency Comparison and Optimizations

Efficiency Comparison. In our first instantiation, in addition to the cost associated with the GC-based ZK, i.e. the oblivious transfer for x and the cost of garbling f, $O(n)$ exponentiations are necessary to commit to each bit of input x and to perform the bitwise ZK proofs associated with them in the last step.

In our second instantiation, the bitwise commitments/proofs are eliminated (i.e. only a constant number of exponentiations) but instead the circuit for $ax+b$ needs to be garbled which requires $O(ns+s^2)$ additional symmetric-key operations when using textbook multiplication (we discuss range of values for s shortly). Using Karatsuba's multiplication algorithm [Knu69], this can potentially be further reduced.

The round complexity of both protocols is essentially the same as the GC-based ZK proof of [JKO13] (5 rounds), as the extra messages can be sent within the same rounds. (To simplify presentation, we used a separate step for each operation in our protocol description, but many of these can be combined.) A more round-efficient GC-based ZK proof would make our constructions more round efficient as well.

The first instantiation requires more exponentiations which are significantly costlier than their symmetric-key counterpart, but the total number of symmetric-key operations in the second instantiation is higher. Hence, when n is small, the first instantiation is likely more efficient, while when n is larger, the second instantiation will be the better option. Furthermore, if bit-wise commitment to the input is already necessary as part of the bigger protocol (as is the case in some of our constructions), the first instantiation may be the better choice. In the case where a comparison circuit $x < q$ is also computed, an additional $O(n)$ symmetric-key operations suffices.

Optimizations. Next, we review a few other optimizations that improve efficiency of our instantiations.

- *Reducing exponentiations.* We consider the following optimization for the protocol $\Pi_{\mathsf{Com},f}$ in Fig. 3.2 which reduces the number of exponentiations

necessary for the ZK proofs significantly. In step 6, the prover commits to the sum of the keys received instead of individually to each wire key. The prover sends $\mathsf{Com}(S) = \mathsf{Com}\left(\sum_{i=1}^{n} X'_i\right)$ in step 6. We assume that the bit commitment scheme Com is homomorphic, and each wire key X_i is truncated to s bits and interpreted as a group element. Now, in the zero knowledge proofs of step 14, the prover proves the following statements which can be performed with fewer exponentiations:

$$\mathsf{PK}\{(x_i, S, r, R) : \mathsf{Com}(x_i) = g^{x_i} h^r \wedge \mathsf{Com}(S) = g^S h^R \\ \wedge S = \sum_{i=1}^{n} \left(x_i X_i^1 + (1 - x_i) X_i^0\right)\}$$

$$\mathsf{PK}\{(x, x_1, \cdots, x_n, r, r_1, \cdots r_n) : \mathsf{Com}(x) = g^x h^r \wedge \mathsf{Com}(x_i) = g^{x_i} h^{r_i} \\ \wedge x = g^{\sum 2^i x_i} h^r\}$$

We can show that if the sum extracted by the simulator from the commitment in step 6 is not equal to the sum of keys corresponding to the input x' extracted from COT, but the ZK proofs verify, then for some i, the prover must have correctly guessed X_i^b such that $b \neq x'_i$. The probability of this is negligible by the security of the COT protocol.

- *Privacy-free garbling.* As discussed earlier, in [FNO15] it is observed that privacy-free garbling is sufficient for GC-based ZK proofs of non-algebraic statements. This improves the communication/computation cost of garbled circuits in our first instantiation by a factor of two. But as mentioned earlier, the same cannot be said about our second construction since the privacy property of garbling is required to hide a and b in the earlier stage of the construction.

  But we can think of bigger circuit as consisting of two smaller circuits: one computing the function f and the other computing $ax + b$. If we split the computation into two garbled circuits with shared OT, then we can use the privacy free garbling scheme of [FNO15, ZRE15] for the first circuit as the verifier has no input, and use a standard garbling scheme for the $ax + b$ circuit.

- *Smaller multiplication circuit.* For the one-time MAC in the second protocol, a small a is sufficient for security - if the security (unforgeability) desired is 2^{-s}, it suffices for a to be s bits long. Hence, for a 512-bit input, a $40-80$-bit a is sufficient to compute $ax + b$ which reduces the size of the multiplication circuit significantly.

3.2 Sigma Protocols and SNARKs for Combination Statements

We begin by presenting a sigma protocol to prove equality of committed values with the exponents a_i in $y = \prod G_i^{a_i}$. Then, we show how to prove consistency of the input and output of a circuit in a SNARK with values committed to in algebraic commitments.

3.2.1 Proof of equality of committed values

Let $\mathbb{G}$ be a group of prime order q. Let $y = \prod G_i^{a_i}, C_i = g^{a_i} h^{r_i}$, where g, G_i, h are generators of the group, and the prover does not know the discrete logarithms of h with respect to g. We want to prove equality of the discrete logarithms in y and the respective values committed to in C_i. Let k be a security parameter. Following standard notation, we denote the protocol by $\mathsf{PK}\{(a_1, \cdots, a_n, r_1, \cdots, r_n) : y = \prod_{i=1}^{n} G_i^{a_i} \wedge C_i = g^{a_i} h^{r_i}\}$.

Figure 3.4: The Protocol comEq

Given $y = \prod_{i=1}^{n} G_i^{a_i}$ and $C_i = g^{a_i} h^{r_i}$

1. The prover computes the following values: $u = \prod_{i=1}^{n} G_i^{\alpha_i}$ and $v_i = g^{\alpha_i} h^{R_i}$ for randomly chosen $\alpha_i, R_i \in \mathbb{Z}_q$ and sends u, v_i to the verifier.
2. The verifier chooses a random string c of length k as the challenge, and sends it to the prover.
3. For a challenge string c, compute and send the tuple (s_i, t_i)

$$s_i = \alpha_i - c a_i \pmod{q}, t_i = R_i - c r_i \pmod{q}$$

4. Verification:

   Check if $u = y^c \prod G_i^{s_i}$ and $v_i = (C_i)^c g^{s_i} h^{t_i}$. The verifier accepts if Verification succeeds for all i.

We will show that the protocol in Figure 3.4 is correct, has a soundness error of $1/2^k$, and is honest verifier zero knowledge.

Proof. • **Completeness:**

If the prover and the verifier behave honestly, it is easy to see that verification conditions hold.

$$y^c \prod G_i^{s_i} = (\prod G_i^{a_i})^c \prod G_i^{s_i} = \prod G_i^{a_i c + s_i} = u$$

$$(C_i)^c g^{s_i} h^{t_i} = (g^{a_i} h^{r_i})^c g^{s_i} h^{t_i} = g^{ca_i + s_i} h^{cr_i + t_i} = v_i$$

- **Soundness:** We show an extractor that computes $a_1, \cdots a_n, r_1, \cdots, r_n$ given two accepting views with same commitments but different challenge strings. Say, we have two accepting views: $\{(u, v_i), c, (s_i, t_i)\}$ and $\{(u, v_i), \hat{c}, (\hat{s}_i, \hat{t}_i)\}$ for challenges c and $\hat{c} \neq c$. Since the views are accepting, we have,

$$y^c \prod G_i^{s_i} = y^{\hat{c}} \prod G_i^{\hat{s}_i} = u$$
$$y^{c-\hat{c}} = \prod G_i^{\hat{s}_i - s_i}$$

  We can now compute (in $\mathbb{Z}_q$), $x_i = (\hat{s}_i - s_i)(c - \hat{c})^{-1}$. The inverse of $(c - \hat{c})$ exists in $\mathbb{Z}_q$, since $c \neq \hat{c}$ by assumption.

  Similarly,

$$(C_i)^c g^{s_i} h^{t_i} = (C_i)^{\hat{c}} g^{\hat{s}_i} h^{\hat{t}_i} = v_i$$

  and we can compute

$$r_i = (\hat{t}_i - t_i)(c - \hat{c})^{-1}$$

  The extractor succeeds in extracting a witness given two accepting transcripts. The prover can, therefore, cheat only when he can answer exactly one challenge correctly, and the probability of that challenge being chosen by the verifier is bounded by $1/2^k$ where k is the length of the challenge.

- **Honest Verifier Zero Knowledge:** We show a simulator such that the output of the simulator is statistically indistinguishable from the transcript of the protocol with a prover. The simulator on input c, randomly chooses $s_i, t_i \in \mathbb{Z}_q$ and computes $u = y^c \prod G_i^{s_i}$, and $v_i = (C_i)^c g^{s_i} h^{t_i}$.

□

3.2.2 SNARK on committed input

The starting point of our constructions is the verifiable computation protocol of Pinocchio. At a high level, each polynomial of the quadratic program (defined in 2.5), say, $v_k(x) \in \mathbb{F}$ is mapped to an element in a bilinear group, $g^{v_k(s)}$, where s

is a secret value chosen during CRS generation, g is a generator of the group. $\mathbb{F}$ is the field of discrete logarithms and over which the operations of the computation are performed. Given these group elements and the values a_i on the circuit wires which are the coefficients of the quadratic program, the prover can compute "in the exponent" to obtain $g^{v(s)}$, where $v(s) = \sum a_i v_k(s)$. The verifier uses the bilinear map to verify that the divisibility check of the QAP holds. We assume the computations are over large fields, that is, the QAP is defined over $\mathbb{F}_p$ for a large p. The size of the field is exponential in the security parameter. We omit p in all further descriptions of the field.

Let $f : \mathbb{F}^N \to \mathbb{F}^{n'}$ be a function with input/output values from $\mathbb{F}$, computed by an arithmetic circuit C with input wires labeled $1, \ldots, N$, output wires labeled $m - n' + 1, \ldots, m$. Let $\mathcal{Q}$ be a QAP of size m and degree d corresponding to C. We separate the circuit wires I into private input, public input, intermediate values, and output wires. Let $I_{com} \subseteq \{1, \cdots, N\}$ be the set of indices corresponding to the private inputs $x_1, \cdots, x_n$, I_{pub} the indices for the public input wires, and I_{out} the indices for the public output. Let $I_{mid} = \{1, \cdots, m\} \setminus I_{pub} \cup I_{com} \cup I_{out}$. Let C_i be an algebraic commitment, for example, Pedersen, to the ith input $x_i, C_i = g^{x_i} h^{r_i}$. The construction $\mathsf{comInSnark} : \mathsf{PK}\{(x_1, \cdots, x_n, r_1, \cdots, r_n) : f(x_1, \cdots x_n, z_1, \cdots, z_{N-n}) = (b_1, \cdots, b_{n'}) \wedge C_1 = g^{x_1} h^{r_1} \wedge \cdots \wedge C_n = g^{x_n} h^{r_n}\}$ is given in Fig. 3.5.

Given $C_i = g^{x_i} h^{r_i}$ for all $i \in [n]$, commitments to private inputs, the public inputs, $z_1, \cdots, z_{N-n}$, and the public outputs, $b_1, \cdots, b_{n'}$. Let g be a generator of $\mathbb{G}_1$, $\tilde{g}$ a generator of $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, a non-trivial bilinear map.

1. CRS generation: Choose $r_v, r_w, \alpha_v, \alpha_w, \alpha_y, s, \beta, \gamma \xleftarrow{R} \mathbb{F}$. Set $r_y = r_v r_w, g_v = g^{r_v}, g_w = g^{r_w}, \tilde{g}_w = \tilde{g}^{r_w}, g_y = g^{r_y}$.

   Set the CRS to be:

$$\begin{aligned}\mathsf{crs} = (&\{g_v^{v_k(s)}\}_{k\in I_{com}}, \{g_v^{v_k(s)}\}_{k\in I_{mid}}, \{\tilde{g}_w^{w_k(s)}\}_{k\in I_{com}}, \{\tilde{g}_w^{w_k(s)}\}_{k\in I_{mid}},\\ &\{g_y^{y_k(s)}\}_{k\in I_{com}}, \{g_y^{y_k(s)}\}_{k\in I_{mid}} \{g_v^{\alpha_v v_k(s)}\}_{k\in I_{com}}, \{g_v^{\alpha_v v_k(s)}\}_{k\in I_{mid}},\\ &\{g_w^{\alpha_w w_k(s)}\}_{k\in I_{com}}, \{g_w^{\alpha_w w_k(s)}\}_{k\in I_{mid}}, \{g_y^{\alpha_y y_k(s)}\}_{k\in I_{com}}, \{g_y^{\alpha_y y_k(s)}\}_{k\in I_{mid}}\\ &\{g^{s^i}\}_{i\in[d]}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k\in I_{com}}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k\in I_{mid}})\end{aligned}$$

   Set the short verification CRS to be:

$$\begin{aligned}\mathsf{shortcrs} = (&g, \tilde{g}, \tilde{g}^{\alpha_v}, g^{\alpha_w}, \tilde{g}^{\alpha_y}, \tilde{g}^{\gamma}, g^{\beta\gamma}, \tilde{g}^{\beta\gamma}, g^{t(s)},\\ &\{g_v^{v_k(s)}\}_{k\in I_{pub}\cup I_{out}}, \{\tilde{g}_w^{w_k(s)}\}_{k\in I_{pub}\cup I_{out}}, \{g_y^{y_k(s)}\}_{k\in I_{pub}\cup I_{out}})\end{aligned}$$

2. Prove:

On input $z_1, \cdots, z_{N-n}$, witness $x_1, \cdots, x_n$, and crs, the prover evaluates the QAP to obtain $\{a_i\}_{i\in[m]}$. (Equivalently, evaluates the circuit to obtain the values on the circuit wires). The prover solves for the quotient polynomial h such that $p(x) = h(x)t(x)$. Let $v_{com}(x) = \sum_{k\in I_{com}} a_k v_k(x)$, $v_{mid}(x) = \sum_{k\in I_{mid}} a_k v_k(x)$ and similarly define $w_{com}(x)$, $w_{mid}(x)$, $y_{com}(x)$ and $y_{mid}(x)$.

- The prover computes the proof π:

$$(g_v^{v_{com}(s)}, g_v^{v_{mid}(s)}, \tilde{g}_w^{w_{com}(s)}, \tilde{g}_w^{w_{mid}(s)}, g_y^{y_{com}(s)}, g_y^{y_{mid}(s)}, \tilde{g}^{h(s)},$$
$$g_v^{\alpha_v v_{com}(s)}, g_v^{\alpha_v v_{mid}(s)}, g_w^{\alpha_w w_{com}(s)}, g_w^{\alpha_w w_{mid}(s)}, g_y^{\alpha_y y_{com}(s)}, g_y^{\alpha_y y_{mid}(s)}$$
$$g_v^{\beta v_{com}(s)} g_w^{\beta w_{com}(s)} g_y^{\beta y_{com}(s)}, g_v^{\beta v_{mid}(s)} g_w^{\beta w_{mid}(s)} g_y^{\beta y_{mid}(s)})$$

- Prove input consistency with commitment: The prover uses the sigma protocol comEq to compute π_{in}: $\mathsf{PK}\{(x_1, \cdots, x_n, r_1, \cdots, r_n) : y = \prod_{i=1}^n G_i^{x_i} \wedge C_1 = g^{x_1} h^{r_1} \wedge \cdots \wedge C_n = g^{x_n} h^{r_n}\}$, for $G_i = g_v^{v_i(s)}, i \in I_{com}$, and $y = g_v^{v_{com}(s)}$

3. Verify:

- On input $\mathsf{shortcrs}$, z, and proofs π, π_{in} parse π as

$$\pi = (g^{V_{com}}, g^{V_{mid}}, \tilde{g}^{W_{com}}, \tilde{g}^{W_{mid}}, g^{Y_{com}}, g^{Y_{mid}}, \tilde{g}^{H},$$
$$g^{V'_{com}}, g^{V'_{mid}}, g^{W'_{com}}, g^{W'_{mid}}, g^{Y'_{com}}, g^{Y'_{mid}}, g^{Z_{com}}, g^{Z_{mid}})$$

- Divisibility check. Compute $g_v^{v_{io}(s)} = \prod_{k\in I_{pub}\cup I_{out}} (g_v^{v_k(s)})^{a_k}$. Similarly, compute $\tilde{g}_w^{w_{io}(s)}$ and $g_y^{y_{io}(s)}$.

$$e(g_v^{v_0(s)} g_v^{v_{io}(s)} g^{V_{com}} g^{V_{mid}}, \tilde{g}_w^{w_0(s)} \tilde{g}_w^{w_{io}(s)} \tilde{g}^{W_{com}} \tilde{g}^{W_{mid}})$$
$$= e(g^{t(s)}, \tilde{g}^{H}) e(g_y^{y_0(s)} g_y^{y_{io}(s)} g^{Y_{com}} g^{Y_{mid}}, \tilde{g})$$

- Verify that the linear combinations are in correct spans.
  (a) $e(g^{V'_{com}}, \tilde{g}) = e(g^{V_{com}}, \tilde{g}^{\alpha_v})$
  (b) $e(g^{V'_{mid}}, \tilde{g}) = e(g^{V_{mid}}, \tilde{g}^{\alpha_v})$
  (c) $e(g^{W'_{com}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{com}})$
  (d) $e(g^{W'_{mid}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{mid}},)$
  (e) $e(g^{Y'_{com}}, \tilde{g}) = e(g^{Y_{com}}, \tilde{g}^{\alpha_y})$
  (f) $e(g^{Y'_{mid}}, \tilde{g}) = e(g^{Y_{mid}}, \tilde{g}^{\alpha_y})$
- Verify same coefficients in all linear combinations.
  (a) $e(g^{Z_{com}}, \tilde{g}^{\gamma}) = e(g^{V_{com}} g^{Y_{com}}, \tilde{g}^{\beta\gamma}) e(g^{\beta\gamma}, \tilde{g}^{W_{com}})$

(b) $e(g^{Z_{mid}}, \tilde{g}^{\gamma}) = e(g^{V_{mid}} g^{Y_{mid}}, \tilde{g}^{\beta\gamma}) e(g^{\beta\gamma}, \tilde{g}^{W_{mid}})$

- Verify input consistency with commitment: The verifier computes $G_i = g_v^{v_i(s)}, i \in I_{com}$, and sets $y = g^{V_{com}}$. The verifier checks that the comEq proof π_{in} verifies: $\mathsf{PK}\{(x_1, \cdots, x_n, r_1, \cdots, r_n) : y = \prod_{i=1}^{n} G_i^{x_i} \wedge C_1 = g^{x_1} h^{r_1} \wedge \cdots \wedge C_n = g^{x_n} h^{r_n}\}$

Figure 3.5: The Protocol comInSnark

Zero-knowledge. We make our construction zero-knowledge, and obtain zk-comInSnark, by randomizing the elements in the proof π such that the checks verify and the proof is statistically indistinguishable from random group elements. Specifically, the prover chooses random $\delta_{v_{com}}, \delta_{v_{mid}}, \delta_{w_{com}}, \delta_{w_{mid}} \leftarrow \mathbb{F}$, and adds $\delta_{v_{com}} t(s)$ in the exponent to $v_{com}(s)$, $\delta_{v_{mid}} t(s)$ to $v_{mid}(s)$, $\delta_{w_{com}} t(s)$ to $w_{com}(s)$, and $\delta_{w_{mid}} t(s)$ to $w_{mid}(s)$. It is easy to see that the modified value of $p(x)$ remains divisible by $t(x)$. The above makes the proof statistically zero-knowledge. To allow the prover to randomize the proof, the following terms are added to the crs: $g_v^{t(s)}, \tilde{g}_w^{t(s)}, g_y^{t(s)}, g_v^{\alpha_v t(s)}, g_w^{\alpha_w t(s)}, g_y^{\alpha_y t(s)}, g_v^{\beta t(s)}, g_w^{\beta t(s)}, g_y^{\beta t(s)}$. The new values in π can now be computed from the crs. Verification proceeds as before. For the input consistency with commitment step: The verifier computes $G_i = g_v^{v_i(s)}, i \in I_{com}, H = g_v^{t(s)}$, and sets $y = g^{V_{com}}$. The proof π_{in}: $\mathsf{PK}\{(a_1, \cdots, a_n, \delta, r_1, \cdots, r_n) : y = H^{\delta} \prod_{i=1}^{n} G_i^{a_i} \wedge C_1 = g^{a_1} h^{r_1} \wedge \cdots \wedge C_n = g^{a_n} h^{r_n}\}$ is statistically zero-knowledge.

We recall a technical lemma from [GGPR13] below, on which we rely for soundness.

Lemma 3.2.1 (Lemma 10, [GGPR13])**.** *Let $\mathbb{F}[x]^{(k)}$ denote polynomials over $\mathbb{F}[x]$ of degree at most k. Let $\mathbb{F}[x]^{(\neg k)}$ denote polynomials over $\mathbb{F}[x]$ that have a zero coefficient for x^k. For some d, let $\mathcal{U} = \{u_k(x)\} \subset \mathbb{F}[x]^d$, and let $\mathsf{span}(\mathcal{U})$ denote the set of polynomials that can be generated as $\mathbb{F}$-linear combinations of the polynomials in $\mathcal{U}$. Let $a(x) \in \mathbb{F}[x]^{(d+1)}$ be generated uniformly at random subject to the constraint that $\{a(x) \cdot u_k(x) : u_k(x) \in \mathcal{U}\} \subset \mathbb{F}[x]^{(\neg(d+1))}$. Let $s \in \mathbb{F}^*$. Then, for all algorithms $\mathcal{A}$*

$$\Pr[u(x) \leftarrow \mathcal{A}(\mathcal{U}, s, a(s)) : u(x) \in \mathbb{F}[x]^d \wedge u(x) \notin \mathsf{span}(\mathcal{U}) \wedge a(x) \cdot u(x) \in \mathbb{F}[x]^{(\neg(d+1))}] \leq \frac{1}{|\mathbb{F}|}$$

Theorem 3.2.2. *If the q-PDH, 2q-SDH and d-PKE assumptions hold for $q \geq 4d + 4$, then zk-comInSnark instantiated with a QAP of degree d is secure under definition 2.2.4 with soundness error $1/|\mathbb{F}|$.*

Proof. **Soundness.**

Assume there exists an adversary $\mathcal{A}$ who returns the proof of a false statement. We use this adversary $\mathcal{A}$ along with the knowledge extractor that exists by the d-PKE assumption to construct an adversary $\mathcal{B}$ to break either the q-PDH assumption or the $2q$-SDH assumption. $\mathcal{B}$ is given the challenge $g, \tilde{g}, g^s, \tilde{g}^s, \ldots, g^{s^q}, \tilde{g}^{s^q}$, $g^{s^{q+2}}, \tilde{g}^{s^{q+2}}, \cdots, g^{s^{2q}}, \tilde{g}^{s^{2q}}$. $\mathcal{A}$ generates a function f that has a QAP $\mathcal{Q} = (t(x), \mathcal{V}, \mathcal{W}, \mathcal{Y})$ of size m and degree d. $\mathcal{B}$ sets the CRS and short CRS according to the protocol, and choosing the randomness in the following way. $\mathcal{B}$ chooses $r_v, r_w, \alpha_v, \alpha_w, \alpha_y$ at random and sets $r_y = r_v r_w, g_v = g^{r_v s^{d+1}}, g_w = g^{r_w s^{2(d+1)}}, g_y = g^{r_y s^{3(d+1)}}$

β is set to be a polynomial evaluated at s in the following way. Rewriting the final term in the proof π, we have,

$$g_v^{\beta v(s)} g_w^{\beta w(s)} g_y^{\beta y(s)} = g^{\beta(r_v s^{d+1} v(s) + r_w s^{2(d+1)} w(s) + r_y s^{3(d+1)} y(s))} \tag{3.1}$$

$\mathcal{B}$ sets $\beta = s^{q-(4d+3)} \beta_{poly}(s)$ where, $\beta_{poly}(x)$ is a polynomial of degree at most $3d+3$ sampled uniformly at random such that $\beta_{poly}(x) \cdot (r_v v_k(x) + r_w x^{(d+1)} w_k(x) + r_y x^{2(d+1)} y_k(x))$ has a zero coefficient in front of x^{3d+3} for all k. Such a polynomial is guaranteed to exist by Lemma 3.2.1. Rewriting equation 3.1 by writing β in terms of s,

$$g_v^{\beta v(s)} g_w^{\beta w(s)} g_y^{\beta y(s)} = g^{\beta(r_v s^{d+1} v(s) + r_w s^{2(d+1)} w(s) + r_y s^{3(d+1)} y(s))} \tag{3.2}$$

$$= g^{s^{q-3d-2} r_v \beta_{poly}(s) v(s) + s^{q-2d-1} r_w \beta_{poly}(s) w(s) + s^{q-d} r_y \beta_{poly}(s) y(s)} \tag{3.3}$$

$$= g^{s^{q-3d-2} \beta_{poly}(s)(r_v v(s) + s^{d+1} r_w w(s) + s^{2d+2} r_y y(s))} \tag{3.4}$$

Since $\beta_{poly}(x) \cdot (r_v v_k(x) + r_w x^{(d+1)} w_k(x) + r_y x^{2(d+1)} y_k(x))$ has a zero coefficient in front of x^{3d+3}, the exponent in equation 3.2 has a zero in front of s^{q+1}. The powers of q in the exponent go up to $(q - 3d - 2) + (3d + 3) + (2d + 2) + d = q + 3d + 3 \leq 2q$. $\mathcal{B}$ can efficiently generate the terms in the CRS that contain β by using the elements in the challenge. $\mathcal{B}$ generates γ' uniformly at random from $\mathbb{F}$ and sets $\gamma = \gamma' s^{q+2}$. $\mathcal{B}$ can generate g^γ from the challenge, since $g^{s^{q+2}}$ is given. Note, $\beta\gamma = s^{q-(4d+3)} \beta_{poly}(s) \gamma' s^{q+2}$ does not have the s^{q+1} term, and has degree at most $q - (4d + 3) + (3d + 3) + (q + 2) \leq 2q$. Hence, $\mathcal{B}$ can generate $g^{\beta\gamma}$ using the elements in its challenge. The polynomials $v_k(x), w_k(x), y_k(x)$ are of degree d, and since we have $q \geq 4d + 4$, all the elements in the CRS can be generated using terms in the challenge.

Let $(\widehat{\pi}, \widehat{\pi_{in}})$ be a cheating proof returned by $\mathcal{A}$ for the computation of f with public input and public output $\{c_k\}_{k \in I_{pub} \cup I_{out}}$. Let $\widehat{\pi} = (g^{V_{com}}, g^{V_{mid}}, \tilde{g}^{W_{com}}, \tilde{g}^{W_{mid}}, g^{Y_{com}}, g^{Y_{mid}}, \tilde{g}^H, g^{V'_{com}}, g^{V'_{mid}}, g^{W'_{com}}, g^{W'_{mid}}, g^{Y'_{com}}, g^{Y'_{mid}}, g^{Z_{com}}, g^{Z_{mid}})$. Since the verification holds, we have that $e(g^{V'_{com}}, \tilde{g}) = e(g^{V_{com}}, \tilde{g}^{\alpha_v})$, and $e(g^{V'_{mid}}, \tilde{g}) = e(g^{V_{mid}}, \tilde{g}^{\alpha_v})$. $\mathcal{B}$ can run the d-PKE extractor to recover polynomials $V_{mid}(x)$ and $V_{com}(x)$ of degree at most d such that $V_{mid} = V_{mid}(s), V_{com} = V_{com}(s)$. Note that the parameters received by $\mathcal{A}$ is a valid input for the d-PKE assumption from which all

the other terms in the CRS can be efficiently generated. That is, the d-PKE adversary receives input (σ, z), where $\sigma = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g}, \{g^{s^i}\}_{i\in[0,d]}, \{\tilde{g}^{s^i}\}_{i\in[0,d]})$, and the auxiliary input z consists of all the other terms in the CRS. Note that the terms $\{g_v^{v_k(s)}\}$ can be efficiently generated from σ. $\mathcal{A}$ returns (V, V') such that $e(V, \tilde{g}) = e(V', \tilde{g}^{\alpha_v})$. Thus, $\mathcal{B}$ can invoke the d-PKE extractor χ_A to recover a polynomial $V_{com}(x) = \sum_{i=0}^{d} c_i x^i$ of degree at most d such that $V = g^{V_{com}(s)}$. Similarly, $\mathcal{B}$ recovers polynomials $W_{mid}(x), W_{com}(x), Y_{mid}(x), Y_{com}(x)$ such that $W_{mid} = W_{mid}(s), W_{com} = W_{com}(s), Y_{mid} = Y_{mid}(s), Y_{com} = Y_{com}(s)$. Now, $\mathcal{B}$ computes,

$$V(x) = v_0(x) + \sum_{k\in I_{pub}} c_k v_k(x) + \sum_{k\in I_{out}} c_k v_k(x) + V_{com}(x) + V_{mid}(x)$$

and similarly $W(x)$ and $Y(x)$, and sets $H(x) = (V(x)W(x) - Y(x))/t(x)$

Since the proof is of a false statement, either the extracted polynomials do not form a QAP solution, or the co-efficients of the extracted *com* polynomials are not equal to the values committed to in $C_1, \cdots, C_n$. There are the following cases:

- $H(x)$ has a non-trivial denominator.
- The polynomial $R(x) = r_v x^{d+1} V_{mid}(x) + r_w x^{2(d+1)} W_{mid}(x) + r_y x^{3(d+1)} Y_{mid}(x)$ is not in the linear subspace generated by the polynomials $\{r_k(x) = r_v x^{d+1} v_k(x) + r_w x^{2(d+1)} w_k(x) + r_y x^{3(d+1)} y_k(x)\}_{k\in I_{mid}}$
- The polynomial $S(x) = r_v x^{d+1} V_{com}(x) + r_w x^{2(d+1)} W_{com}(x) + r_y x^{3(d+1)} Y_{com}(x)$ is not in the linear subspace generated by the polynomials $\{r_k(x) = r_v x^{d+1} v_k(x) + r_w x^{2(d+1)} w_k(x) + r_y x^{3(d+1)} y_k(x)\}_{k\in I_{com}}$
- By the soundness of the protocol comEq, there exists an extractor that extracts $a_1, \cdots, a_n$ such that $V_{com}(s) = \sum_{k\in I_{com}} a_k v_k(s), C_i = g^{a_i} h^{r_i}$. $a_1, \cdots, a_n$ are different from the coefficients c_i of the polynomial V_{com} extracted by the d-PKE extractor.

If none of the above cases hold, then $V(x), W(x), Y(x)$ are a QAP solution, with input consistent with commitments C_i.

Case 1 $t(x)$ does not divide $p(x) = V(x)W(x) - Y(x)$. Let $(x - r)$ be a polynomial that divides $t(x)$ but not $p(x)$, and let $T(x) = t(x)/(x - r)$. Let $d(x) = \gcd(t(x), p(x))$. $t(x)$ has degree at most d and $p(x)$ has degree at most $2d$. $\mathcal{B}$ can use the extended Euclidean algorithm to find polynomials $a(x), b(x)$ with degrees $2d - 1$ and $d - 1$ respectively, such that $a(x)t(x) + b(x)p(x) = d(x)$. Now set $A(x) = a(x) \cdot (T(x)/d(x))$ and $B(x) = b(x) \cdot (T(x)/d(x))$. $A(x)$

and $B(x)$ do not have any denominator since $d(x)$ divides $T(x)$. We have, $A(x)t(x) + B(x)p(x) = T(x)$. Dividing by $t(x)$ we have, $A(x) + B(x)H(x) = \frac{1}{(x-r)}$. $A(x)$ and $B(x)$ have degree at most $2d-1 \leq q$; hence, $\mathcal{B}$ can use the terms in its challenge to compute $e(g^{A(s)}, \tilde{g})e(g^{B(s)}, \tilde{g}^H) = e(g, \tilde{g})^{1/(s-r)}$ which solves the $2q$-SDH.

Case 2 There does not exist $\{c_k\}_{k \in I_{mid}}$ such that $V_{mid}(x) = \sum_{k \in I_{mid}} c_k v_k(x), W_{mid}(x) = \sum_{k \in I_{mid}} c_k w_k(x)$ and $Y_{mid}(x) = \sum_{k \in I_{mid}} c_k y_k(x)$. By Lemma 3.2.1, we have that $x^{q-(4d+3)}\beta_{poly}(x)(r_v x^{d+1} v_k(x) + r_w x^{2(d+1)} w_k(x) + r_y x^{3(d+1)} y_k(x))$ has a non-zero coefficient for the x^{q+1} term with high probability. $\mathcal{B}$ can use $g^{Z_{mid}} = g^{s^{q-(4d+3)}\beta_{poly}(x)(s^{d+1}V_{mid}(s) + s^{2(d+1)}W_{mid}(s) + s^{3(d+1)}Y_{mid}(s))}$ to subtract off all elements of the form g^{s^j} for $j \neq q+1$, and obtain $g^{s^{q+1}}$. This breaks the q-PDH assumption.

Case 3 Similar to Case 2 with V_{com} polynomial, and using $g^{Z_{com}}$.

Case 4 This breaks the binding property of the multi-commitment y, since we have $y = \prod_{i \in I_{com}} G_i^{a_i} = \prod_{i \in I_{com}} G_i^{c_i}$, $a_i \neq c_i$ for some $i \in I_{com}$.

Zero-knowledge. We now show a simulator (S, Sim) such that S outputs a simulated crs and trapdoor, and Sim outputs a simulated proof. S generates crs in the same way and sets the trapdoor τ to be $\tau = (s, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma)$. Sim, given the trapdoor τ picks polynomials $v(x), w(x)$ at random such that $t(x)$ divides $v(x)w(x)$. It sets $h(x)$ to be the quotient polynomial. Now, it chooses polynomials $v_{com}(x), w_{com}(x)$ at random, and sets $v_{mid}(x) = v(x) - v_0(x) - v_{io}(x) - v_{com}(x)$, and $w_{mid}(x) = w(x) - w_0(x) - w_{io}(x) - w_{com}(x)$. Given these polynomials, and s, α, β, γ from the trapdoor, Sim can compute the encodings of $V_{mid} = v_{mid}(s), V_{com} = v_{com}(s)$, and other elements of the proof. Moreover, the simulated proof elements are statistically uniform, subject to the verification constraints. By the zero-knowledge property of the protocol comEq, there exists a simulator that is invoked by Sim to generate a simulated proof that is statistically indistinguishable from π_{in}.

□

3.2.3 SNARK on committed input/output

We separate the circuit wires into private input, private output, intermediate values and private output. Let $I_{com} \subseteq \{1, \cdots, m\}$ be the set of indices corresponding to the private inputs $x_1, \cdots, x_n$, and I_{pub} the indices for the public input wires. Let I_{out} be the set of indices corresponding to the outputs b_i,

and $I_{mid} = \{1, \cdots, m\} \setminus I_{pub} \cup I_{com} \cup I_{out}$. Let C_i be an algebraic commitment, for example, Pedersen, to the ith input $x_i, C_i = g^{x_i} h^{r_i}$, and D_i, commitment to the outputs $D_i = g^{b_i} h^{R_i}$. The construction $\mathsf{comIOSnark} : \mathsf{PK}\{(x_1, \cdots, x_n, b_1, \cdots, b_{n'}, r_1, \cdots, r_n, R_1, \cdots, R_{n'}) : f(x_1, \cdots x_n, z) = (b_1, \cdots, b_{n'}) \wedge C_i = g^{x_i} h^{r_i} \wedge D_i = g^{b_i} h^{R_i}\}$ is given in Fig. 3.6.

Given $C_i = g^{x_i} h^{r_i}$, for all $i \in [n]$, commitments to private inputs, $D_i = g^{b_i} h^{R_i}$, for all $i \in [n']$,commitments to private outputs, and public input z. Let g be a generator of $\mathbb{G}_1$, $\tilde{g}$ a generator of $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, a non-trivial bilinear map.

1. CRS generation:

   - Choose $r_v, r_w, \alpha_v, \alpha_w, \alpha_y, s, \beta, \gamma \xleftarrow{R} \mathbb{F}$. Set $r_y = r_v r_w, g_v = g^{r_v}, g_w = g^{r_w}, \tilde{g}_w = \tilde{g}^{r_w}, g_y = g^{r_y}$.
   - Set the CRS to be:

$$\begin{aligned}\mathsf{crs} = (&\{g_v^{v_k(s)}\}_{k\in I_{com}}, \{g_v^{v_k(s)}\}_{k\in I_{out}}, \{g_v^{v_k(s)}\}_{k\in I_{mid}}, \{\tilde{g}_w^{w_k(s)}\}_{k\in I_{com}},\\ &\{\tilde{g}_w^{w_k(s)}\}_{k\in I_{out}}, \{\tilde{g}_w^{w_k(s)}\}_{k\in I_{mid}}, \{g_y^{y_k(s)}\}_{k\in I_{com}}, \{g_y^{y_k(s)}\}_{k\in I_{out}},\\ &\{g_y^{y_k(s)}\}_{k\in I_{mid}}\{g_v^{\alpha_v v_k(s)}\}_{k\in I_{com}}, \{g_v^{\alpha_v v_k(s)}\}_{k\in I_{out}}, \{g_v^{\alpha_v v_k(s)}\}_{k\in I_{mid}},\\ &\{g_w^{\alpha_w w_k(s)}\}_{k\in I_{com}}, \{g_w^{\alpha_w w_k(s)}\}_{k\in I_{out}}, \{g_w^{\alpha_w w_k(s)}\}_{k\in I_{mid}}, \{g_y^{\alpha_y y_k(s)}\}_{k\in I_{com}},\\ &\{g_y^{\alpha_y y_k(s)}\}_{k\in I_{out}}, \{g_y^{\alpha_y y_k(s)}\}_{k\in I_{mid}}\{g^{s^i}\}_{i\in[d]}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k\in I_{com}},\\ &\{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k\in I_{out}}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k\in I_{mid}})\end{aligned}$$

   - Set the short verification CRS to be:

$$\begin{aligned}\mathsf{shortcrs} = (&g, \tilde{g}, \tilde{g}^{\alpha_v}, g^{\alpha_w}, \tilde{g}^{\alpha_y}, \tilde{g}^{\gamma}, g^{\beta\gamma}, \tilde{g}^{\beta\gamma}, g_y^{t(s)},\\ &\{g_v^{v_k(s)}\}_{k\in I_{pub}}, \{\tilde{g}_w^{w_k(s)}\}_{k\in I_{pub}}, \{g_y^{y_k(s)}\}_{k\in I_{pub}})\end{aligned}$$

2. Prove. On input z, witness $x_1, \cdots, x_n, b_1, \cdots, b_{n'}$, and crs, the prover evaluates the QAP to obtain $\{a_i\}_{i\in[m]}$. (Equivalently, evaluates the circuit to obtain the values on the circuit wires). The prover solves for the quotient polynomial h such that $p(x) = h(x)t(x)$. Let $v_{com}(x) = \sum_{k\in I_{com}} a_k v_k(x)$, $v_{mid}(x) = \sum_{k\in I_{mid}} a_k v_k(x)$, $v_{out}(x) = \sum_{k\in I_{out}} a_k v_k(x)$ and similarly define $w_{com}(x)$, $w_{mid}(x)$, $w_{out}(x)$, $y_{com}(x)$, $y_{mid}(x)$ and $y_{out}(x)$.

- The prover computes the proof π:

$$(g_v^{v_{com}(s)}, g_v^{v_{mid}(s)}, g_v^{v_{out}(s)}, \tilde{g}_w^{w_{com}(s)}, \tilde{g}_w^{w_{mid}(s)}, \tilde{g}_w^{w_{out}(s)}, g_y^{y_{com}(s)},$$
$$g_y^{y_{mid}(s)}, g_y^{y_{out}(s)}, \tilde{g}^{h(s)}, g_v^{\alpha_v v_{com}(s)}, g_v^{\alpha_v v_{mid}(s)}, g_v^{\alpha_v v_{out}(s)}, g_w^{\alpha_w w_{com}(s)},$$
$$g_w^{\alpha_w w_{mid}(s)}, g_w^{\alpha_w w_{out}(s)}, g_y^{\alpha_y y_{com}(s)}, g_y^{\alpha_y y_{mid}(s)}, g_y^{\alpha_y y_{out}(s)},$$
$$g_v^{\beta v_{com}(s)} g_w^{\beta w_{com}(s)} g_y^{\beta y_{com}(s)}, g_v^{\beta v_{mid}(s)} g_w^{\beta w_{mid}(s)} g_y^{\beta y_{mid}(s)},$$
$$g_v^{\beta v_{out}(s)} g_w^{\beta w_{out}(s)} g_y^{\beta y_{out}(s)})$$

- Prove input consistency with commitment. The prover uses sigma protocol comEq to compute proof π_{in}: $\mathsf{PK}\{(x_1, \cdots, x_n, r_1, \cdots, r_n) : y = \prod_{i=1}^{n} G_i^{x_i} \wedge C_1 = g^{x_1} h^{r_1} \wedge \cdots \wedge C_n = g^{x_n} h^{r_n}\}$, for $G_i = g_v^{v_i(s)}, i \in I_{com}$, and $y = g_v^{v_{com}(s)}$.
- Prove output consistency with commitment. The prover uses sigma protocol comEq to compute proof π_{out}: $\mathsf{PK}\{(b_1, \cdots, b_{n'}, R_1, \cdots, R_{n'}) : y = \prod_{i=1}^{n'} G_{m-n'+i}^{b_i} \wedge D_1 = g^{b_1} h^{R_1} \wedge \cdots \wedge D_{n'} = g^{b_{n'}} h^{R_{n'}}\}$, for $G_j = g_v^{v_j(s)}, j \in I_{out}$, and $y = g_v^{v_{out}(s)}$

3. Verify.

- On input $\mathsf{shortcrs}$, y, and a proof π, parse it as

$$\pi = (g^{V_{com}}, g^{V_{mid}}, g^{V_{out}}, \tilde{g}^{W_{com}}, \tilde{g}^{W_{mid}}, \tilde{g}^{W_{out}}, g^{Y_{com}},$$
$$g^{Y_{mid}}, g^{Y_{out}}, \tilde{g}^{H}, g^{V'_{com}}, g^{V'_{mid}}, g^{V'_{out}}, g^{W'_{com}},$$
$$g^{W'_{mid}}, g^{W'_{out}}, g^{Y'_{com}}, g^{Y'_{mid}}, g^{Y'_{out}}, g^{Z_{com}}, g^{Z_{mid}}, g^{Z_{out}})$$

- Divisibility check. Compute $g_v^{v_{pub}(s)} = \prod_{k \in I_{pub}} (g_v^{v_k(s)})^{a_k}$. Similarly, compute $\tilde{g}_w^{w_{pub}(s)}$ and $g_y^{y_{pub}(s)}$. Check that,
$e(g_v^{v_0(s)} g_v^{v_{pub}(s)} g_v^{V_{com}} g_v^{V_{mid}} g_v^{V_{out}}, \tilde{g}_w^{w_0(s)} \tilde{g}_w^{w_{pub}(s)} \tilde{g}_w^{W_{com}} \tilde{g}_w^{W_{mid}} \tilde{g}_w^{W_{out}})$
$= e(g_y^{t(s)}, \tilde{g}^H) e(g_y^{y_0(s)} g_y^{y_{pub}(s)} g_y^{Y_{com}} g_y^{Y_{mid}} g_y^{Y_{out}}, \tilde{g})$
- Verify that the linear combinations are in correct spans.
  (a) $e(g^{V'_{com}}, \tilde{g}) = e(g^{V_{com}}, \tilde{g}^{\alpha_v})$
  (b) $e(g^{V'_{mid}}, \tilde{g}) = e(g^{V_{mid}}, \tilde{g}^{\alpha_v})$
  (c) $e(g^{V'_{out}}, \tilde{g}) = e(g^{V_{out}}, \tilde{g}^{\alpha_v})$
  (d) $e(g^{W'_{com}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{com}})$
  (e) $e(g^{W'_{mid}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{mid}})$
  (f) $e(g^{W'_{out}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{out}})$
  (g) $e(g_y^{Y'_{com}}, \tilde{g}) = e(g^{Y_{com}}, \tilde{g}^{\alpha_y})$

(h) $e(g_y^{Y'_{mid}}, \tilde{g}) = e(g^{Y_{mid}}, \tilde{g}^{\alpha_y})$

(i) $e(g_y^{Y'_{out}}, \tilde{g}) = e(g^{Y_{out}}, \tilde{g}^{\alpha_y})$

- Verify same coefficients in all linear combinations.
  (a) $e(g^{Z_{com}}, \tilde{g}^{\gamma}) = e(g^{V_{com}} g^{Y_{com}}, \tilde{g}^{\beta\gamma}) e(g^{\beta\gamma}, \tilde{g}^{W_{com}})$
  (b) $e(g^{Z_{mid}}, \tilde{g}^{\gamma}) = e(g^{V_{mid}} g^{Y_{mid}}, \tilde{g}^{\beta\gamma}) e(g^{\beta\gamma}, \tilde{g}^{W_{mid}})$
  (c) $e(g^{Z_{out}}, \tilde{g}^{\gamma}) = e(g^{V_{out}} g^{Y_{out}}, \tilde{g}^{\beta\gamma}) e(g^{\beta\gamma}, \tilde{g}^{W_{out}})$
- Verify input consistency with commitment. Verify comEq proof π_{in}. The verifier computes $G_i = g_v^{v_i(s)}, i \in I_{com}$, and sets $y = g^{V_{com}}$ from the proof π. The verifier checks that the proof π_{in} is a proof of knowledge of: $\mathsf{PK}\{(x_1, \cdots, x_n, r_1, \cdots, r_n) : y = \prod_{i=1}^{n} G_i^{x_i} \wedge C_1 = g^{x_1} h^{r_1} \wedge \cdots \wedge C_n = g^{x_n} h^{r_n}\}$.
- Verify output consistency with commitment. Verify comEq proof π_{out}. The verifier computes $G_i = g_v^{v_i(s)}, i \in I_{out}$, and sets $y = g^{V_{out}}$ from the proof π. The verifier checks that the proof π_{out} verifies. $\mathsf{PK}\{(b_1, \cdots, b_{n'}, R_1, \cdots, R_{n'}) : y = \prod_{i=1}^{n'} G_{m-n'+i}^{b_i} \wedge D_1 = g^{b_1} h^{R_1} \wedge \cdots \wedge D_{n'} = g^{b_{n'}} h^{R_{n'}}\}$.

Figure 3.6: The Protocol comIOSnark

3.2.4 Sigma protocols on committed outputs

In [CDS94], the authors devise an OR composition technique for sigma protocols. Essentially, a prover can efficiently show $((x_0 \in \mathcal{L}) \vee (x_1 \in \mathcal{L}))$, without revealing which x_i is in the language. We show how to use the OR composition to construct a sigma protocol with committed output. In particular, given algebraic commitments to inputs $x_1, \cdots, x_n$, public $y_1, \cdots, y_m$ and an efficient sigma protocol to prove that $f(x_1, \cdots, x_n, y_1, \cdots, y_m) = 1$, we show how to construct an efficient sigma protocol to prove $f(x_1, \cdots, x_n, y_1, \cdots, y_m) = b$, for a committed bit b. Let C_i be a commitment to the ith input x_i. $\mathsf{PK}\{(b, x_1, \cdots, x_n) : f(x_1, \cdots, x_n, y_1, \cdots, y_m) = b \wedge D_b = g^b h^R \wedge C_i = g^{x_i} h^{r_i}\}$

- The prover commits to the output bit $b, D_b = g^b h^R$
- The prover proves the following OR statement:

$$\mathsf{PK}\{(b, x_1, \cdots, x_n) : \big(f(x_1, \cdots, x_n, y_1, \cdots, y_m) = 1 \wedge b = 1 \wedge D_b = g^b h^R \wedge C_i = g^{x_i} h^{r_i}\big) \vee \big(b = 0 \wedge D_b = g^b h^R\big)\}$$

Chapter 4

Applications of ZK for Combination Statements[1]

In this chapter, we show how to use the techniques to combine garbled circuits and SNARKs with sigma protocols from Chapter 3 in applications. We focus on anonymous credentials and proof of solvency as application examples, and begin by discussing some necessary building blocks.

4.1 Building Blocks for Privacy-Preserving Signature Verification

We now look at anonymous credentials as an application for our zero-knowledge proofs for combination statements. We use the constructions from Chapter 3 to base credentials on standard signatures. We introduce three important building blocks for our privacy-preserving signature verification protocols. Two of them can be directly instantiated using our $\mathcal{F}_{\mathsf{Com},f}$ functionality introduced in Section 3.1, while for the third one we provide a customized construction.

4.1.1 Proving that a committed value is the hash of another committed value

Here, the goal is to commit to a message m and its hash $\mathcal{H}(m)$ and prove in zero-knowledge that one committed value is the hash of the other. We define the task in terms of the ideal functionality in Figure 4.1.

[1] This chapter is based on joint work with Melissa Chase and Payman Mohassel that appeared in CRYPTO 2016 [CGM16], and joint work with Shashank Agrawal and Payman Mohassel that is yet to be published [AGM17]. Some passages are taken verbatim from these sources.

Figure 4.1: The ideal functionality $\mathcal{F}_{Hash}$

- The verifier inputs $\mathsf{Com}(m), \mathsf{Com}(M)$ and the prover inputs the opening information (m, M) and the randomness.
- If $\mathcal{H}(m) = M$ and the openings to the commitments verify, output accept to the verifier.

We now use the abstract functionality $\mathcal{F}_{\mathsf{Com},f}$ from Figure 3.1 with a commitment scheme Com_h to instantiate a protocol that implements $\mathcal{F}_{Hash}$. Here, the input is $x = (m, M = \mathcal{H}(m))$ and the Com_h is defined as $\mathsf{Com}_h(x = (m, M)) = (\mathsf{Com}(m), \mathsf{Com}(M))$. Recall, one of our protocols (Section 3.1.4), required bitwise commitments from the prover. To commit to bits of x, one can commit to bits of m and M individually. Com_h inherits efficient proofs of linear relations from Com as long as the proofs on m and M are performed separately. Given these, we show in Figure 4.2 how to implement $\mathcal{F}_{Hash}$ by defining the right function f for the ideal functionality $\mathcal{F}_{\mathsf{Com},f}$.

Figure 4.2: The Protocol Π_{Hash}

1. The prover commits to $x = (m, M)$ by sending $\mathsf{Com}_h(x) = \mathsf{Com}(m), \mathsf{Com}(M)$ to the verifier.
2. The prover and the verifier run $\Pi_{\mathsf{Com},f}$ where f is the following functionality: f takes m and M as inputs and outputs v such that $v = 1$ if $\mathcal{H}(m) = M$ and 0 otherwise.

Theorem 4.1.1. *The protocol Π_{Hash} in Figure 4.2 securely implements $\mathcal{F}_{Hash}$, given the ideal functionality $\mathcal{F}_{\mathsf{Com},f}$, in the presence of malicious adversaries.*

4.1.2 Proof of equality of committed values in different groups

The goal is to prove that the value committed to in different prime groups of size p and q are the same. We define the task in terms of an ideal functionality, defined in Figure 4.3. This can be achieved using standard techniques which involve using the integer commitment scheme by Damgard and Fujisaki [DF02] to prove properties about the discrete logarithms in $\mathbb{Z}$ (instead of modulo the order of the group). This requires that the verifier choose an RSA modulus $\widetilde{N}$ such that the factorization is unknown to the prover, and prove that it is chosen correctly in

an initial set-up phase. The prover also has to compute exponentiations in an RSA group where the exponents are $|\widetilde{N}| + \kappa$ bits long. Since the group order is hidden, chinese remaindering cannot be used to speed up the exponentiations, and therefore the approach is fairly expensive. We give a protocol that avoids the integer commitment technique.

Figure 4.3: The ideal functionality $\mathcal{F}_{Eq}$

- The verifier inputs $\mathsf{Com}_p(x), \mathsf{Com}_q(y)$ and the prover inputs (x, y) and the opening information. p and q are public primes and $q < p$.
- If $0 \leq x < p, 0 \leq y < p, x \equiv y \mod q$, and the openings to the commitments verify, output accept to the verifier.

In Figure 4.4, we use the ideal functionality $\mathcal{F}_{\mathsf{Com},f}$ from Figure 3.1 with a commitment scheme Com_{pq} to instantiate a protocol that implements $\mathcal{F}_{Eq}$. The scheme is defined as $\mathsf{Com}_{pq}(x) = (\mathsf{Com}_p(x), \mathsf{Com}_q(x))$, where it is assumed that Com_p and Com_q allow for proving linear relationships among committed values.

Figure 4.4: The Protocol Π_{Eq}

1. The prover commits to x and y by sending $\mathsf{Com}_p(x), \mathsf{Com}_q(y)$ to the verifier.
2. The prover and the verifier run $\Pi_{\mathsf{Com},f}$ where f is the following functionality: f takes x and checks that it is upper bounded by p and outputs v such that $v = 1$ if $x \leq p$ and 0 otherwise.

4.1.3 Proof of equality of discrete logarithm of a committed value and another committed value

Let $\mathbb{G}_1 = \langle G_1 \rangle$ and $\mathbb{G}_2 = \langle G_2 \rangle$ be two groups of order p and q respectively with $q|(p-1)$ and let $g \in \mathbb{G}_2$ be an element of order q. Given $y_1 = G_1^{g^x} H_1^{R_1}$ and $y_2 = G_2^x H_2^{R_2}$, we want to prove that the discrete logarithm w.r.t to base g of the value committed to in y_1 is equal to the value committed to in y_2. Let k be a security parameter. Following standard notation, we denote the protocol by $\mathsf{PK}\{(x, R_1, R_2) : y_1 = G_1^{g^x} H_1^{R_1} \wedge y_2 = G_2^x H_2^{R_2}\}$. The technique of our protocol is similar to [Sta96], [CS97a], and is a variant of [MGGR13]. Our protocol is only honest verifier zero-knowledge. This HVZK protocol can be compiled into a full zero-knowledge proof of knowledge in the auxiliary string model using the

technique of [Dam00]. The protocol $\mathsf{PK}\{(x, R_1, R_2) : y_1 = G_1^{g^x} H_1^{R_1} \wedge y_2 = G_2^x H_2^{R_2}\}$ is given in Figure 4.5.

Figure 4.5: Double discrete logarithm proof

Given $y_1 = G_1^{g^x} H_1^{R_1}$ and $y_2 = G_2^x H_2^{R_2}$

1. The prover computes the following $2k$ values: $u_i = G_1^{g^{\alpha_i}} H_1^{\beta_i}$ and $v_i = G_2^{\alpha_i} H_2^{\gamma_i}$ for $1 \leq i \leq k$, for randomly chosen $\alpha_i, \gamma_i \in \mathbb{Z}_q$ and $\beta_i \in \mathbb{Z}_p$, and sends u_i, v_i to the verifier.

2. The verifier chooses a random string c of length k as the challenge, and sends it to the prover.

3. For a challenge string $c = c_1 \ldots c_k$, compute and send the tuple (r_i, s_i, t_i)
   If $c_i = 0$,
   $$r_i = \alpha_i, s_i = \beta_i, t_i = \gamma_i$$
   If $c_i = 1$,
   $$r_i = \alpha_i - x \pmod q, s_i = \beta_i - R_1 g^{r_i} \pmod p, t_i = \gamma_i - R_2 \pmod q$$

4. Verification:
   If $c_i = 0$, check whether $u_i = G_1^{g^{r_i}} H_1^{s_i}$ and $v_i = G_2^{r_i} H_2^{t_i}$
   If $c_i = 1$, check if $u_i = y_1^{g^{r_i}} H_1^{s_i}$ and $v_i = y_2 G_2^{r_i} H_2^{t_i}$. The verifier accepts if Verification succeeds for all i.

We will show that the protocol in Figure 4.5 is correct, has a soundness error of $1/2^k$, and is honest verifier zero knowledge.

Proof. • **Completeness:**

If the prover and the verifier behave honestly, it is easy to see that verification conditions hold:
If $c_i = 0$:

$$G_1^{g_i^r} H_1^{s_i} = G_1^{g^{\alpha_i}} H_1^{\beta_i} = u_i \text{ and } G_2^{r_i} H_2^{t_i} = G_2^{\alpha_i} H_2^{\gamma_i} = v_i$$

If $c_i = 1$:

$$y_1^{g^{r_i}} H_1^{s_i} = (G_1^{g^x})^{g^{r_i}} (H_1^{R_1})^{g^{r_i}} H_1^{s_i} = G_1^{g^{\alpha_i}} H_1^{\beta_i} = u_i \text{ and },$$

$$y_2 G_2^{r_i} H_2^{t_i} = G_2^x H_2^{R_2} G_2^{r_i} H_2^{t_i} = v_i$$

- **Soundness:** We show an extractor that computes x, R_1, R_2 given two different accepting views with same commitments but different challenge strings. Say, we have two accepting views for challenges c and $\hat{c} \neq c$. Without loss of generality, let us assume that they differ in the jth position, and $c_j = 0$. We have,

$$u_j = G_1^{g^{r_j}} H_1^{s_j} = y_1^{g^{\hat{r_j}}} H_1^{\hat{s}_j}$$

$$G_1^{g^{r_j}} H_1^{s_j} = G_1^{g^x g^{\hat{r_j}}} H_1^{Rg^{\hat{r_j}} + \hat{s_j}}$$

$$g^x = g^{r_j - \hat{r_j}}$$

We can compute (in $\mathbb{Z}_q$),

$$x = r_j - \hat{r_j}$$

We have,

$$s_j = R_1 g^{\hat{r_j}} + \hat{s_j}$$

and thus,

$$R_1 = \frac{s_j - \hat{s_j}}{g^{\hat{r_j}}}$$

We also have

$$v_j = G_2^{r_j} H_2^{t_j} = y_2 G_2^{\hat{r_j}} H_2^{\hat{t_j}}$$

$$G_2^{r_j} H_2^{t_j} = G_2^{x + \hat{r_j}} H_2^{\hat{t_j} + R_2}$$

and thus,

$$R_2 = t_j - \hat{t_j}$$

- **Honest Verifier Zero Knowledge:** We show a simulator such that the output of the simulator is statistically indistinguishable from the transcript of the protocol with a prover. The simulator on input c, randomly chooses $\alpha_i = r_i \in \mathbb{Z}_q, \beta_i = s_i \in \mathbb{Z}_p, \gamma_i = t_i \in \mathbb{Z}_q$ and computes for $1 \leq i \leq k$:
If $c_i = 0$,

$$u_i = G_1^{g^{r_i}} H_1^{s_i} \text{ and } v_i = G_2^{r_i} H_2^{t_i}$$

if $c_i = 1$,

$$u_i = y_1^{g^{r_i}} H_1^{s_i} \text{ and } v_i = y_2 G_2^{r_i} H_2^{t_i}$$

□

4.2 Privacy-Preserving Signature Verification

4.2.1 RSA signatures

The FDH-RSA Scheme. The Full Domain Hash RSA signature scheme $\mathsf{FDH} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is defined as follows [BR93a]. The KeyGen algorithm on input the security parameter k, selects two $k/2$-bit primes p and q and computes the modulus $N = pq$. It then chooses an exponent $e \in \mathbb{Z}^*_{\phi(N)}$, and computes d such that $ed = 1 \mod \phi(N)$. Return (pk, sk), where $pk = (N, e)$ and $sk = (N, d)$. The signature generation and verification are as follows and use a hash function $\mathcal{H} : \{0,1\} \to \mathbb{Z}^*_N$.

$\mathsf{Sign}_{N,d}(M)$	$\mathsf{Verify}_{N,e}(M, \sigma)$
$x = \mathcal{H}(M)$	$y = \sigma^e \mod N$
$\sigma = x^d \mod N$	$y' = \mathcal{H}(M)$
return σ	**if** $(y = y')$ **then return** 1;
	else return 0;

4.2.1.1 Proof of Knowledge of RSA Signatures

Given $\mathsf{Com}_N(m)$, a commitment to m in a group of order N, the following protocol is a zero knowledge proof of knowledge of a valid RSA signature on m.

1. The prover has input (m, σ) and the verifier is in possession of $\mathsf{Com}_N(m) = C_1 = g^m h^{r_1}$
2. The prover commits to $M = \mathcal{H}(m)$, that is, $M \in \mathbb{Z}_N$, compute $\mathsf{Com}_N(M) = C_2 = g^M h^{r_2}$, for randomly chosen $r_2 \in Z^*_N$. Send C_2 to the verifier and prove knowledge of opening.
3. The prover and verifier engage in the protocol Π_{Hash} with inputs (m, M) and (C_1, C_2) respectively.
4. The prover proves knowledge of e-th root of a committed value [CS97a]. Given $y = C_2 = g^M h^r$, prover proves knowledge of σ, such that, $y = g^{\sigma^e} h^r$.

   (a) The prover computes the following tuple:

   $$(y_1, \cdots, y_{e-1}) \text{ where } y_i = g^{\sigma^i} h^{r_i}$$

   for randomly chosen $r_i \in \mathbb{Z}_N$, for $i = 1$ to $e - 1$.

(b) The prover and the verifier run the following proof of knowledge:

$$\mathsf{PK}\{(\alpha, (\beta_1, \cdots, \beta_e)) : y_1 = g^{\alpha} h^{\beta_1} \wedge y_2 = y_1^{\alpha} h^{\beta_2} \wedge \cdots \wedge y = y_{e-1}^{\alpha} h^{\beta_e}\}$$

When e is one greater than a power of 2, we can employ optimizations like repeated squaring to prove knowledge of e-th root. Given $y = g^{\sigma^e} h^r$, for $e = 2^k + 1$, step 4 in the verification protocol can be now be realized as follows:

1. The prover computes the following tuple:

$$(y_0, y_1, \cdots, y_k) \text{ where } y_i = g^{\sigma^{2^i}} h^{r_i}$$

for randomly chosen $r_i \in \mathbb{Z}_N$, for $i = 1$ to k.

2. The prover and the verifier run the following proof of knowledge:

$$\begin{aligned}\mathsf{PK}\{(\alpha, \alpha_1, \cdots, \alpha_k, \beta, \beta_0, \cdots, \beta_k, R_0, \cdots, R_k) :& \\ y_0 = g^{\alpha} h^{\beta} \wedge y_1 = y_0^{\alpha} h^{\beta_0} \wedge y_1 = g^{\alpha_1} h^{R_0} \wedge y_2 = y_1^{\alpha_1} h^{\beta_1} & \\ \wedge\, y_2 = g^{\alpha_2} h^{R_1} \cdots \wedge y_k = y_{k-1}^{\alpha_{k-1}} h^{\beta_{k-1}} \wedge y_k = g^{\alpha_k} h^{R_{k-1}} \wedge y = y_k^{\alpha} h^{\beta_k}\}&\end{aligned}$$

It might be possible to improve the efficiency for some e's by using addition chains for the integer e. An addition chain for integer e is an ascending sequence $1 = e_0 < e_1 < \cdots e_r = e$ such that for each i, $1 \leqslant i \leqslant r$, there is some j and k with $1 \leqslant j \leqslant k < i$ and $e_i = e_j + e_k$. The prover, now, would have to provide only the y_i's for which i is an element of the addition chain for e. The relations among the y_i's will be sightly different, but can be proved in a similar way.

The above verification protocol can also be adapted to support variants of RSA-based signatures, like the probabilistic signature scheme (PSS) from [BR96]. PSS is a probabilistic generalization of FDH which uses two hash functions and more complicated padding. We can instantiate protocol $\Pi_{\mathsf{Com},f}$ with an f that verifies the additional checks of PSS to achieve privacy preserving verification of a PSS signature.

Proof of security. We sketch a proof that the above protocol is a zero-knowledge proof of knowledge of an RSA signature on a committed message.

1. Completeness: By correctness of the protocol Π_{Hash}, we have that $M = \mathcal{H}(m)$. We now show the completeness of the proof in step 4. We have that the interactive protocol corresponding to $\mathsf{PK}\{(\alpha, (\beta_1, \cdots, \beta_e)) : y_1 = g^{\alpha} h^{\beta_1} \wedge y_2 = y_1^{\alpha} h^{\beta_2} \wedge \cdots \wedge y = y_{e-1}^{\alpha} h^{\beta_e}\}$ is a proof of knowledge of values $\alpha, \beta_1, \cdots, \beta_e$. It follows that,

$$y = \left(y_{e-1}^{\alpha}\right) h^{\beta_e} = \left(\left(\cdots \left(g^{\alpha} h^{\beta_1}\right)^{\alpha} h^{\beta_2} \cdots\right)^{\alpha} h^{\beta_{e-1}}\right)^{\alpha} h^{\beta_e}$$

$$= g^{\alpha^e} h^{\beta_e + \alpha\beta_{e-1} + \cdots + \alpha^{e-1}\beta_1}$$

If the prover and the verifier behave honestly, the verifier accepts.

2. Soundness: We show an extractor, that, given access to the prover, extracts (m, σ) such that $\mathsf{Verify}_{N,e}(m, \sigma) = 1$. The extractor invokes the simulator for the corrupt prover of protocol Π_{Hash} to extract m and M. It then runs the extractor corresponding to the proof in step 4b to extract α. By the security of Π_{Hash} and the binding property of Com, it follows that $\alpha^e \mod N = M = \mathcal{H}(m)$.

3. Zero-knowledge: We sketch a simulator that simulates the verifier's view in the protocol. The simulator commits to a random value on behalf of the prover in step 2 by computing $C_2' = \mathsf{Com}(M')$. It sends C_2' to the verifier, proves knowledge of opening and invokes the simulator for the corrupt verifier of protocol Π_{Hash}. It then chooses $y_1, \cdots, y_{e-1} \in Z_N$ at random, and runs the simulator corresponding to the proof in step 4b. We can show that the view of the verifier in the protocol is indistinguishable from the view with the simulator via a sequence of intermediate games.

   - Game $\mathcal{G}_0$: This is the real game of the verifier with the honest prover.
   - Game $\mathcal{G}_1$: This game is similar to $\mathcal{G}_0$ except in step 3, the simulator for protocol Π_{Hash} is invoked instead of honestly running Π_{Hash}.
     Games $\mathcal{G}_0$ and $\mathcal{G}_1$ are indistinguishable by the security of Π_{Hash}.
   - Game $\mathcal{G}_2$: This game is similar to game $\mathcal{G}_1$ above except that in step 4b, the simulator for the zero-knowledge proof is invoked instead of doing the proof honestly.
     Games $\mathcal{G}_1$ and $\mathcal{G}_2$ are indistinguishable by the zero-knowledge property of the proof of knowledge.
   - Game $\mathcal{G}_3$: This game is similar to game $\mathcal{G}_2$ except that a random value is committed to in step 2 instead of using the real input.
     Games $\mathcal{G}_2$ and $\mathcal{G}_3$ are indistinguishable by the hiding property of the commitment scheme Com. We also note that $\mathcal{G}_3$ is the interaction of the corrupt verifier with the simulator.

4.2.2 The DSA Scheme.

The Digital Signature Algorithm (DSA) is a variant of the Elgamal signature scheme. The key generation, signature generation and verification algorithms are given next. The KeyGen algorithm chooses two primes p and q such that $q \mid p-1$. Let g be an element of order q in $\mathbb{Z}_p^*$. It then chooses x randomly from $\{1, \cdots, q-1\}$. The private key is set to be x and the public key is $(g, p, q, y), y = g^x \mod p$.

Sign(m)	Verify$(m, (r, s))$
$M \leftarrow \mathcal{H}(m)$	$M \leftarrow \mathcal{H}(m)$
Pick a random $k, 1 \leq k < q$	$w = s^{-1} \mod q$
$r = (g^k \mod p) \mod q$	$u_1 = Mw \mod q$
$s = k^{-1}(M + rx) \mod q$	$u_2 = rw \mod q$
return (r, s)	**if** $r = (g^{u_1} y^{u_2} \mod p) \mod q$ **then return**; 1 **else return** 0;

The ECDSA Scheme. ECDSA is the elliptic curve analogue of DSA. It works in an elliptic curve group $E(\mathbb{Z}_p)$. The ECDSA Key generation, signature and verification algorithms are given below. The KeyGen algorithm chooses an elliptic curve E defined over $\mathbb{Z}_p$ such that the number of points in $E(\mathbb{Z}_p)$ is divisible by a large prime n. Pick a point $P \in E(\mathbb{Z}_p)$ of order n. Let $d \in [1, n-1]$ be a randomly chosen integer. Set $Q = dP$. The public key is (E, P, Q, n) and the private key is d.

Sign(m)	Verify$(m, (r, s))$
$M \leftarrow \mathcal{H}(m)$	$M \leftarrow \mathcal{H}(m)$
Pick a random $k \in [1, n-1]$	**if** $r, s \notin [1, n-1]$ **then return**; 0
$kP = (x_0, y_0)$	$w = s^{-1} \mod n$
$r = x_0 \mod n$	$u_1 = Mw \mod n$
$s = k^{-1}(M + rd) \mod n$	$u_2 = rw \mod n$
return (r, s)	$(x_1, y_1) = u_1 P + u_2 Q$
	$v = x_1 \mod n$
	if $r = v$ **then return** 1; **else return** 0;

4.2.2.1 Proof of Knowledge of DSA Signatures

Let (r, s) be the DSA signature on m. Let $\mathbb{G}_1 = \langle G_1 \rangle$ and $\mathbb{G}_2 = \langle G_2 \rangle$ be two distinct groups of order p and q respectively where p and q are the parameters of the DSA signature algorithm. One technical difficulty is that we have to show r in G_1 and G_2 is equal modulo q. For that purpose, we use our protocol Π_{Eq} from Figure 4.4 to prove equality across groups. We also employ our protocol from Figure 4.5 to prove equality of discrete logarithm of a committed value and another committed value. We now describe the DSA verification protocol in detail. Given a commitment to m, the following protocol is a zero-knowledge proof of knowledge of a valid DSA signature on m.

1. The verifier is in possession of $C_1 = \mathsf{Com}_q(m)$, and the prover has as input message $(m, (r, s))$ and the opening information of C_1 to m.

2. The prover commits to $M = \mathcal{H}(m)$, that is, $M \in \mathbb{Z}_q$, compute $C_2 = \mathsf{Com}_q(M)$ Send C_2 to the verifier and prove knowledge of opening.

3. Now the prover and verifier engage in the protocol Π_{Hash} to prove that $M = \mathcal{H}(m)$.

4. The prover commits to the signature (r, s) by sending $\mathsf{Com}_{pq}(r) = (\mathsf{Com}_p(r), \mathsf{Com}_q(r))$ and $\mathsf{Com}_q(s)$. The prover also commits to the following values: $u_1 = \mathcal{H}(m)s^{-1}, u_2 = rs^{-1}, \alpha = g^{u_1}, \beta = y^{u_2}$, where g is the generator of a cyclic group of order q in $\mathbb{Z}_p^*$ used in DSA signing, and y is the DSA public key. The prover sends $\mathsf{Com}_q(u_1), \mathsf{Com}_q(u_2), \mathsf{Com}_p(\alpha), \mathsf{Com}_p(\beta)$.

5. The prover and the verifier carry out the following Σ-protocol zero-knowledge proofs of knowledge:

   (a) $\mathsf{PK}\{(u_1, R_1, R_2) : \mathsf{Com}_p(\alpha) = G_1^{g^{u_1}} H_1^{R_1} \wedge \mathsf{Com}_q(u_1) = G_2^{u_1} H_1^{R_2}\}$

   (b) $\mathsf{PK}\{(u_2, R_1, R_2) : \mathsf{Com}_p(\beta) = G_1^{y^{u_2}} H_1^{R_1} \wedge \mathsf{Com}_q(u_2) = G_2^{u_2} H_1^{R_2}\}$

   (c) $\mathsf{PK}\{(r, \alpha, \beta, R_1, R_2, R_3) : \mathsf{Com}_p(\beta) = G_1^{\beta} H_1^{R_1} \wedge \mathsf{Com}_p(\alpha) = G_1^{\alpha} H_1^{R_2} \wedge \mathsf{Com}_p(r) = G_1^{r} H_1^{R_3} \wedge r = \alpha\beta\}$

   (d) $\mathsf{PK}\{(M, u_1, s, R_1, R_2, R_3) : \mathsf{Com}_q(M) = G_2^{M} H_2^{R_1} \wedge \mathsf{Com}_q(u_1) = G_2^{u_1} H_2^{R_2} \wedge \mathsf{Com}_q(s) = G_2^{s} H_2^{R_3} \wedge M = u_1 s\}$

   (e) $\mathsf{PK}\{(r, u_2, s, R_1, R_2, R_3) : \mathsf{Com}_q(r) = G_2^{r} H_2^{R_1} \wedge \mathsf{Com}_q(u_2) = G_2^{u_2} H_2^{R_2} \wedge \mathsf{Com}_q(s) = G_2^{s} H_2^{R_3} \wedge r = u_2 s\}$

6. The prover and verifier engage in Π_{Eq} with input $\mathsf{Com}_{pq}(r)$.

Proof of security. We sketch a proof that the above protocol is a zero-knowledge proof of knowledge of a DSA signature (r, s) on a committed message.

1. Completeness: The security of protocol Π_{Hash} ensures that $M = \mathcal{H}(m)$. By completeness of the proofs of knowledge of step 5, we have that the privacy preserving verification protocol between an honest prover and verifier will make the verifier accept.

2. Proof of Knowledge: We show an extractor, that, given access to the prover, extracts $(m, (r, s))$ such that $\mathsf{Verify}(m, (r, s)) = 1$. The extractor invokes the simulator for the corrupt prover of protocol Π_{Hash} to extract m and M and the opening information for C_1.

   It then runs the extractor guaranteed by the proof of knowledge property of the proofs in step 5 to extract $u_1, u_2, \alpha, \beta, s, r$. Finally it returns $(m, (r, s))$ and the opening information. By security of Π_{Hash}, Π_{Eq} and the binding property of the commitment scheme Com, it follows that $r = g^{Ms^{-1}} y^{rs^{-1}}$ and $M = \mathcal{H}(m)$.

3. Zero-knowledge: We sketch a simulator that simulates the verifier's view in the protocol. The simulator commits to a random value on behalf of the prover in step 2 by computing $C_2' = \mathsf{Com}(M')$. It sends C_2' to the verifier, proves knowledge of the opening and invokes the simulator for the corrupt verifier of protocol Π_{Hash}. It then commits to random values in step 4, and runs the simulator corresponding to the proofs of knowledge in step 5. Finally in step 6, the simulator invokes the simulator for protocol Π_{Eq}. We can show that the view of the verifier in the protocol is indistinguishable from the view with the simulator via a sequence of intermediate games.

   - Game $\mathcal{G}_0$: This is the real game of the verifier with the honest prover.
   - Game $\mathcal{G}_1$: This game is similar to $\mathcal{G}_0$ except in step 3, the simulator for protocol Π_{Hash} is invoked instead of honestly running Π_{Hash}.

     Games $\mathcal{G}_0$ and $\mathcal{G}_1$ are indistinguishable by the security of Π_{Hash}.
   - Game $\mathcal{G}_2$: This game is similar to game $\mathcal{G}_1$ above except that in step 5, the simulator for the zero-knowledge proof is invoked instead of doing the proof honestly.

     Games $\mathcal{G}_1$ and $\mathcal{G}_2$ are indistinguishable by the zero-knowledge property of the proofs of knowledge.
   - Game $\mathcal{G}_3$: This game is similar to game $\mathcal{G}_2$ except that the in step 6, the simulator for protocol Π_{Eq} is invoked instead of honestly running Π_{Eq}.

     Games $\mathcal{G}_2$ and $\mathcal{G}_3$ are indistinguishable by the security of Π_{Eq}.

- Game $\mathcal{G}_4$: This game is similar to game $\mathcal{G}_3$ except that random values are committed to in steps 2 and 4 instead of using the real input and real computed values.

  Games $\mathcal{G}_3$ and $\mathcal{G}_4$ are indistinguishable by the hiding property of the commitment scheme Com. We also note that $\mathcal{G}_4$ is the interaction of the corrupt verifier with the simulator.

4.2.2.2 Proof of Knowledge of ECDSA Signatures

Let (r, s) be the ECDSA signature on m. Let $\mathbb{G}_1 = \langle G_1 \rangle$ and $\mathbb{G}_2 = \langle G_2 \rangle$ be two distinct groups of order p and n respectively where p is the order of the field of the curve and n is the order of point P. Addition of elliptic curve points which is the group operation requires arithmetic operations in the underlying finite field $\mathbb{Z}_p$ of the curve E. We use a straight forward variant of the protocol in Fig. 4.5 to prove statements about multiples of an elliptic curve point (elliptic curve analogue of exponentiation) inside commitments.

1. The verifier is in possession of $C_1 = \mathsf{Com}_p(m)$ and the prover has as input (m, σ) and the opening of C_1 to m.

2. The prover commits to $M = \mathcal{H}(m)$, by computing $C_2 = \mathsf{Com}_p(M)$. Send C_2 to the verifier and prove knowledge of opening.

3. The prover and verifier engage in the protocol Π_{Hash} with inputs (m, M) and (C_1, C_2) respectively.

4. The prover commits to the signature (r, s) and proves knowledge of an opening. The prover sends $\mathsf{Com}_{pn}(r) = (\mathsf{Com}_p(r), \mathsf{Com}_n(r))$ and $\mathsf{Com}_n(s)$. The prover also commits to the following values: $u_1 = \mathcal{H}(m)s^{-1}, u_2 = rs^{-1}$, and the co-ordinates of the points $u_1P = (\alpha_x, \alpha_y), u_2Q = (\beta_x, \beta_y)$, where P is the point of order n in $E(\mathbb{Z}_p)$ used in ECDSA signing, and Q is the ECDSA public key. The prover sends $\mathsf{Com}_n(u_1)$, $\mathsf{Com}_n(u_2)$, $\mathsf{Com}_p(\alpha_x)$,$\mathsf{Com}_p(\alpha_y)$, $\mathsf{Com}_p(\beta_x)$, $\mathsf{Com}_p(\beta_y)$.

5. The prover and the verifier carry out the following Σ-protocol zero-knowledge proofs of knowledge:

   (a) $\mathsf{PK}\{(u_1, \alpha_x, \alpha_y, R_1, R_2, R_3) : \mathsf{Com}_p(\alpha_x) = G_1^{\alpha_x} H_1^{R_1} \wedge$
   $\mathsf{Com}_p(\alpha_y) = G_1^{\alpha_y} H_1^{R_2} \wedge \mathsf{Com}_n(u_1) = G_2^{u_1} H_1^{R_3} \wedge (\alpha_x, \alpha_y) = u_1 P\}$

   (b) $\mathsf{PK}\{(u_2, \beta_x, \beta_y, R_1, R_2, R_3) : \mathsf{Com}_p(\beta_x) = G_1^{\beta_x} H_1^{R_1} \wedge$
   $\mathsf{Com}_p(\beta_y) = G_1^{\beta_y} H_1^{R_2} \wedge \mathsf{Com}_n(u_2) = G_2^{u_2} H_1^{R_3} \wedge (\beta_x, \beta_y) = u_2 Q\}$

(c) $\mathsf{PK}\{(r, \alpha_x, \alpha_y, \beta_x, \beta_y, R_1, R_2, R_3, R_4, R_5) : \mathsf{Com}_p(\beta_x) = G_1^{\beta_x} H_1^{R_1} \wedge$
$\mathsf{Com}_p(\beta_y) = G_1^{\beta_y} H_1^{R_2} \wedge \mathsf{Com}_p(\alpha_x) = G_1^{\alpha_x} H_1^{R_3} \wedge \mathsf{Com}_p(\alpha_y) =$
$G_1^{\alpha_y} H_1^{R_4} \wedge \mathsf{Com}_p(r) = G_1^{r} H_1^{R_5} \wedge r = ((\alpha_x, \alpha_y) + (\beta_x, \beta_y))_x\}$

(d) $\mathsf{PK}\{(M, u_1, s, R_1, R_2, R_3) : \mathsf{Com}_n(M) = G_2^{M} H_2^{R_1} \wedge$
$\mathsf{Com}_n(u_1) = G_2^{u_1} H_2^{R_2} \wedge \mathsf{Com}_n(s) = G_2^{s} H_2^{R_3} \wedge M = u_1 s\}$

(e) $\mathsf{PK}\{(r, u_2, s, R_1, R_2, R_3) : \mathsf{Com}_n(r) = G_2^{r} H_2^{R_1} \wedge \mathsf{Com}_n(u_2) = G_2^{u_2} H_2^{R_2}$
$\wedge \mathsf{Com}_n(s) = G_2^{s} H_2^{R_3} \wedge r = u_2 s\}$

6. The prover and verifier engage in Π_{Eq} with input $\mathsf{Com}_{pn}(r)$.

The above protocol is a zero knowledge proof of knowledge of ECDSA signature, the proofs for correctness, soundness and zero-knowledge are similar to the proofs of the protocol for the DSA signature.

4.3 Secure computation on committed/signed inputs

In the protocols described above, we have shown how to commit to a value $\mathsf{Com}(x)$ and then use a GC-based ZK proof to prove non-algebraic statements about x.

It is not hard to show that one can extend this approach, to a full-fledged secure two-party computation (2PC) of any function $g(x, y)$ where x is the committed input of the prover. In particular, note that in the ZK proof, the prover feeds its input x into the COTs in order to obtain its inputs keys to the GC of the ZK proof. In order to extend this to a secure 2PC based on garbled circuits, we let the prover play the role of the evaluator in a cut-and-choose 2PC based on garbled circuits, and use the same COT as above for the prover to obtain the garbled inputs for x in the 2PC. This would ensure that the same x that was used in the ZK proof is also used in the 2PC, and the ZK proof already ensures that this is the same input committed to in $\mathsf{Com}(x)$.

A subtle point here is that we need to open the sender's input to the COTs for the GC for the ZK but not for the GCs for the 2PC. This is supported by the committing OT of [sS11] (also see the discussion on COTs in [MR13]). It is interesting to explore the use of OT extension in such COTs where some sender inputs are opened while others are not.

We emphasize that the GCs for the 2PC only garble the desired function g, and hence the GC for the ZK proof is not part of any cut-and-choose. However, we note that the above technique is currently limited to the evaluator's input since the OTs for evaluator's input enable an almost-free check of equality of inputs in

the 2PC and the ZK. Extending the ideas to both party's inputs is an interesting future direction.

This approach can be easily extended to prove other statements about x, such as proof of knowledge of a signature on x (hence signed-input 2PC) either using the techniques we give below in the case of RSA/DSA signatures, or using previous techniques to give a proof of knowledge of a CL signature [CL01].

4.4 Building Blocks for Privacy-Preserving Proof of Solvency

In this section, we construct a proof for committing to g^x where g is a generator for an elliptic curve group, and proving knowledge of x such that $\mathsf{Com}(g^x) = y$ for a public y. Previous techniques for proving such statements are limited to integer groups and are hence are not immediately applicable in applications like protocols for Bitcoin which uses elliptic curve groups. We then use this construction along with our technique to combine SNARKs with sigma protocols from Section 3.2 to build a privacy-preserving proof of solvency for Bitcoin.

4.4.1 Proof of Knowledge of Double Discrete Logarithm in Elliptic Curve Groups

The goal is to prove the equality of a committed value and the discrete logarithm of another committed value. When the commitments are in elliptic curve groups, the known techniques for double discrete logarithm proofs will not work. This is because a group element cannot be naturally interpreted as a field element, as can be done in integer groups. Towards this end, we first describe a protocol to prove that the sum of two elliptic curve points that are committed to, is another public point on the curve.

Let E be a curve defined over $\mathbb{F}_t$. The point addition relation is defined by the point addition equation specific to the curve. Consider the curve given by,

$$y^2 = x^3 + ax + b \tag{4.1}$$

$a, b \in \mathbb{F}_t$. The curve used by Bitcoin sec256k1 is of the above family, for $a = 0, b = 7$. The point addition for the above curve is : Let $P = (x_1, y_1), Q = (x_2, y_2), P, Q \in E(\mathbb{F}_t)$, $T = (x_3, y_3) = P + Q$ where,

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$$

$$y_3 = \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x_3) - y_1$$

We can prove the above relations for committed x_1, x_2, y_1, y_2 using known Sigma protocol techniques. Since the point addition computation is over $\mathbb{F}_t$, the commitments to the coordinates have to be in a group of order t, which is not necessarily the same as p, the order of the group $E(\mathbb{F}_t)$. The Complex Multiplication (CM) method maybe used to find elliptic curve groups of a specific order, which, however might be inefficient for large orders. In the following, we give a protocol without having to find a group of a given order.

We rewrite the point addition formula.

$$x_3x_2^2+x_3x_1^2+x_1x_2^2+x_2x_1^2+x_1^3+x_2^3+2y_1y_2 = y_2^2+y_1^2+2x_1^2x_2+2x_1x_2^2+2x_1x_2x_3 \quad (4.2)$$

$$x_2y_3 + x_3y_2 + x_2y_1 = x_1y_2 + x_3y_1 + x_1y_3 \quad (4.3)$$

Let L_x and R_x denote the left-hand side, and right-hand side respectively of equation 4.2, and L_y and R_y, of equation 4.3.

$$L_x(x_1, y_1, x_2, y_2) = x_3x_2^2 + x_3x_1^2 + x_1x_2^2 + x_2x_1^2 + x_1^3 + x_2^3 + 2y_1y_2$$

$$R_x(x_1, y_1, x_2, y_2) = y_2^2 + y_1^2 + 2x_1^2x_2 + 2x_1x_2^2 + 2x_1x_2x_3$$

$$L_y(x_1, y_1, x_2, y_2) = x_2y_3 + x_3y_2 + x_2y_1$$

$$R_y(x_1, y_1, x_2, y_2) = x_1y_2 + x_3y_1 + x_1y_3$$

We use sigma protocols to prove that L_x, R_x, L_y and R_y satisfy the above relations using committed intermediate values. Let G_2 be an elliptic group of order q such that $q > 2t^3$, and P', Q', points in G_2. We commit to the coordinates and the intermediate values necessary for the proof in G_2, and since the largest intermediate value in equations 4.2 and 4.3 is cubic, the choice of q ensures there is no reduction when the computation is modulo q. Since all computation on committed values will now be modulo q, and the addition equations are to be computed modulo t, we use division with remainder. We prove equality of L_x and R_x modulo q, divide them by t taking away multiples of t, and prove that the remainders are equal. When used together with appropriate range proofs, we get equality modulo t. There are several known techniques to achieve range proofs [CCs08, Bou00], that is, to prove that $x \in [0, S]$ for a public S and committed x. The protocol pointAddition is given in Figure 4.6.

Given $P = (P_x, P_y), Q = (Q_x, Q_y), T = (T_x, T_y), P, Q, T \in E(\mathbb{F}_t), \mathsf{C}_1 = \mathsf{Com}_q(P_x), \mathsf{C}_2 = \mathsf{Com}_q(P_y), \mathsf{C}_3 = \mathsf{Com}_q(Q_x), \mathsf{C}_4 = \mathsf{Com}_q(Q_y)$, $q > 2t^3$, prove that $T = P + Q$.

1. Let $L_x(P_x, P_y, Q_x, Q_y) = k_1 t + r_1, R_x(P_x, P_y, Q_x, Q_y) = k_1' t + r_1', L_y(P_x, P_y, Q_x, Q_y) = k_2 t + r_2, R_y(P_x, P_y, Q_x, Q_y) = k_2' t + r_2'$, for $k_1, k_1', k_2, k_2' < \frac{q}{t}$ and $r_1, r_1', r_2, r_2' < t$.

   Compute and send commitments $\mathsf{C}_4 = \mathsf{Com}_q(L_x), \mathsf{C}_5 = \mathsf{Com}_q(R_x), \mathsf{C}_6 = \mathsf{Com}_q(L_y), \mathsf{C}_7 = \mathsf{Com}_q(R_y), \mathsf{C}_8 = \mathsf{Com}_q(k_1), \mathsf{C}_9 = \mathsf{Com}_q(r_1), \mathsf{C}_{10} = \mathsf{Com}_q(k_1'), \mathsf{C}_{11} = \mathsf{Com}_q(r_1'), \mathsf{C}_{12} = \mathsf{Com}_q(k_2), \mathsf{C}_{13} = \mathsf{Com}_q(r_2), \mathsf{C}_{14} = \mathsf{Com}_q(k_2'), \mathsf{C}_{15} = \mathsf{Com}_q(r_2')$.

2. The prover proves that $(P_x, P_y), (Q_x, Q_y)$ and (T_x, T_y) satisfy the addition equation for the x-coordinate.

   $\pi_1 : \mathsf{PK}\{(P_x, P_y, Q_x, Q_y, L_x, R_x) : \mathsf{C}_1 = \mathsf{Com}_q(P_x) \wedge \mathsf{C}_2 = \mathsf{Com}_q(P_y) \wedge \mathsf{C}_3 = \mathsf{Com}_q(Q_x) \wedge \mathsf{C}_4 = \mathsf{Com}_q(Q_y) \wedge \mathsf{C}_4 = \mathsf{Com}_q(L_x) \wedge \mathsf{C}_5 = \mathsf{Com}_q(R_x) \wedge L_x = T_x Q_x^2 + T_x P_x^2 + P_x Q_x^2 + Q_x P_x^2 + P_x^3 + P_y^3 + 2P_y y \wedge R_x = Q_y^2 + P_y^2 + 2P_x^2 Q_x + 2P_x Q_x^2 + 2P_x Q_x T_x\}$

3. The prover proves that $(P_x, P_y), (Q_x, Q_y)$ and (T_x, T_y) satisfy the addition equation for the y-coordinate.

   $\pi_2 : \mathsf{PK}\{(P_x, P_y, Q_x, Q_y, L_y, R_y) : a_2 = \mathsf{Com}_q(P_x) \wedge a_3 = \mathsf{Com}_q(P_y) \wedge \mathsf{C}_2 = \mathsf{Com}_q(Q_x) \wedge \mathsf{C}_3 = \mathsf{Com}_q(Q_y) \wedge \mathsf{C}_6 = \mathsf{Com}_q(L_y) \wedge \mathsf{C}_7 = \mathsf{Com}_q(R_y) \wedge L_y = Q_x T_y + T_x Q_y + Q_x P_y \wedge R_y = P_x Q_y + T_x P_y + P_x T_y\}$

4. The prover proves the coordinates are in the correct range by giving the proof.

   $\pi_3 : \mathsf{PK}\{(Q_x, Q_y, P_x, P_y) : \mathsf{C}_2 = \mathsf{Com}_q(Q_x) \wedge \mathsf{C}_3 = \mathsf{Com}_q(Q_y) \wedge a_2 = \mathsf{Com}_q(P_x) \wedge a_3 = \mathsf{Com}_q(P_y) \wedge Q_x < t \wedge Q_y < t \wedge P_x < t \wedge P_y < t\}$

5. The prover proves L_x and R_x are equal modulo t, by dividing each side by t, showing correct range for the quotients and the remainders, and proving the remainders are equal.

   $\pi_4 : \mathsf{PK}\{(L_x, R_x, k_1, k_1', r_1, r_1') : \mathsf{C}_4 = \mathsf{Com}_q(L_x) \wedge \mathsf{C}_5 = \mathsf{Com}_q(R_x) \wedge \mathsf{C}_8 = \mathsf{Com}_q(k_1) \wedge \mathsf{C}_9 = \mathsf{Com}_q(r_1) \wedge \mathsf{C}_{10} = \mathsf{Com}_q(k_1') \wedge \mathsf{C}_{11} = \mathsf{Com}_q(r_1') \wedge L_x = k_1 t + r_1 \wedge R_x = k_1' t + r_1' \wedge r_1 < t \wedge r_1' < t \wedge k_1 < \frac{q}{t} \wedge k_1' < \frac{q}{t} \wedge r_1 - r_1' = 0\}$

6. The prover proves L_y and R_y are equal modulo t, by dividing each side by t, showing correct range for the quotients and the remainders, and proving the remainders are equal.

   $\pi_5 : \mathsf{PK}\{(L_y, R_y, k_2, k_2', r_2, r_2') : \mathsf{C}_6 = \mathsf{Com}_q(L_y) \wedge \mathsf{C}_7 = \mathsf{Com}_q(R_y) \wedge \mathsf{C}_{12} = \mathsf{Com}_q(k_2) \wedge \mathsf{C}_{13} = \mathsf{Com}_q(r_2) \wedge \mathsf{C}_{14} = \mathsf{Com}_q(k_2') \wedge \mathsf{C}_{15} = \mathsf{Com}_q(r_2') \wedge L_y = k_2 t + r_2 \wedge R_y = k_2' t + r_2' \wedge r_2 < t \wedge r_2' < t \wedge k_2 < \frac{q}{t} \wedge k_2' < \frac{q}{t} \wedge r_2 - r_2' = 0\}$

Figure 4.6: $\mathsf{pointAddition} : \mathsf{PK}\{(P = (P_x, P_y), Q = (Q_x, Q_y)) : T = (T_x, T_y) = P + Q \wedge \mathsf{C}_1 = \mathsf{Com}_q(P_x) \wedge \mathsf{C}_2 = \mathsf{Com}_q(P_y) \wedge \mathsf{C}_3 = \mathsf{Com}_q(Q_x) \wedge \mathsf{C}_4 = \mathsf{Com}_q(Q_y)\}$

We show that the protocol pointAddition is honest verifier zero-knowledge, and sound with a soundness error of $\frac{1}{2^k}$, where k is the length of the challenge.

- **Honest verifier zero-knowledge.** The simulator invokes the simulator for the proofs $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5$. Zero-knowledge follows from the zero-knowledge of these proofs.

- **Soundness.** We show an extractor that computes $P = (P_x, P_y), Q = (Q_x, Q_y)$ such that T = P+Q, given two accepting transcripts for two different challenge bits. Say, we have two accepting views for challenge bits c and $\hat{c} \neq c$.

  From the soundness of proofs π_1, π_3, π_4, we can extract P_x, P_y, Q_x, Q_y such that $L_x(P_x, P_y, Q_x, Q_y)$ and $R_x(P_x, P_y, Q_x, Q_y)$ satisfy the following.

  $L_x = k_1 t + r_1 \mod q, R_x = k_1' t + r_1' \mod q$ Now,

$$\begin{aligned} L_x \mod t &= ((k_1 t + r_1) \mod q) \mod t \\ &= (k_1 t + r_1) \quad (\text{ Since } q > t^3, k_1 < q/t, r_1 < t) \\ &= r_1 \mod t \quad (\text{ Since } r_1 < t) \\ &= r_1' \mod t \quad (\text{ Since } r_1 = r_1' \mod q, r_1' < t) \\ &= R_x \mod t \quad (\text{ Since } k_1' < q/t, r_1' < t) \end{aligned}$$

  Similarly, from soundness of π_2, π_3, π_5 we get, $L_y = R_y \mod t$

We note that the above protocol may be modified to prove point addition for a committed point T in the following way. The proofs π_1 and π_2 are on committed coordinates (T_x, T_y), and the range proof π_3 also includes proving the range of coordinates of T. We denote the point addition protocol on all committed points $\mathsf{PK}\{(P = (P_x, P_y), Q = (Q_x, Q_y), T = (T_x, T_y)) : T = P + Q \wedge C_1 = \mathsf{Com}(P) \wedge C_2 = \mathsf{Com}(Q) \wedge C_3 = \mathsf{Com}(T)\}$ by comPointAddition.

We now construct a protocol to prove the equality of a committed value and the discrete logarithm of another committed value using the point addition proof. The double discrete logarithm proof is given in Figure 4.7.

Theorem 4.4.1. *Let $E(\mathbb{F}_t)$ be an elliptic curve given by equation 4.1, and $P \in E$ be an element of prime order p. Then,* ec-ddlog *is a Σ-protocol for the relation $R = \{(P, (\lambda, h)) : h = \lambda P, 0 < \lambda < p\}$.*

Proof. We will show that the protocol ec-ddlog is honest verifier zero-knowledge, and sound with a soundness error of $\frac{1}{2^k}$, where k is the length of the challenge.

Given $\mathsf{C}_1 = \mathsf{Com}_p(\lambda), \mathsf{C}_2 = \mathsf{Com}_q(x), \mathsf{C}_3 = \mathsf{Com}_q(y)$, for $q > 2t^3$, prove that $(x, y) = \lambda P$, where $P \in E$ is an element of prime order p, $0 < \lambda < p$, P', Q', points in G_2 of order q.

1. The prover computes the following values: $a_1 = \mathsf{Com}_p(\alpha) = \alpha P + \beta_1 Q, a_2 = \mathsf{Com}_q(\gamma_1) = \gamma_1 P' + \beta_2 Q', a_3 = \mathsf{Com}_q(\gamma_2) = \gamma_2 P' + \beta_3 Q'$ where $\alpha \in \mathbb{F}_p$ and $(\gamma_1, \gamma_2) = \alpha P$.

   and sends a_1, a_2, a_3 to the verifier.

2. The verifier chooses a random challenge bit c and sends it to the prover.

3. For challenge c,

   - If $c = 0$, compute $z_1 = \alpha, z_2 = \beta_1, z_3 = \beta_2, z_4 = \beta_3$. Send the tuple (z_1, z_2, z_3, z_4)
   - If $c = 1$, compute $z_1 = \alpha - \lambda$. Let $T = z_1 P = (t_1, t_2)$. The prover uses pointAddition (Figure 4.6) to prove that $T = (\gamma_1, \gamma_2) - (x, y)$.
     $\pi : \mathsf{PK}\{(x, y, \gamma_1, \gamma_2) : T = (\gamma_1, \gamma_2) - (x, y)\}$. Send (z_1, π)

4. Verification:
   Compute $(t_1, t_2) = z_1 P$. If $c = 0$, check if $a_1 = z_1 P + z_2 Q, a_2 = t_1 P' + z_3 Q', a_3 = t_2 P' + z_4 Q'$.

   If $c = 1$, Verify proof π.

Figure 4.7: ec-ddlog : $\mathsf{PK}\{(\lambda, x, y, r, r_1, r_2) : \mathsf{Com}_p(\lambda) = \lambda P + rQ \wedge \mathsf{Com}_q(x) = xP' + r_1 Q' \wedge \mathsf{Com}_q(y) = yP' + r_2 Q' \wedge (x, y) = \lambda P\}$

- **Honest verifier zero-knowledge.** We can construct a simulator such that the output of the simulator is statistically indistinguishable from the transcript of the protocol with a prover. On input a bit c, the simulator does the following: if $c = 0$, the simulator randomly chooses $z_1, z_2 \in \mathbb{Z}_p, z_3, z_4 \in \mathbb{Z}_q$, and computes $a_1 = z_1P + z_2Q, a_2 = \gamma_1P' + z_3Q', a_3 = \gamma_2P' + z_4Q'$ for $(\gamma_1, \gamma_2) = z_1P$. It is easy to see that the output of the simulator is distributed identically with the distribution of the protocol transcript. If $c = 1$, the simulator randomly chooses $z_1 \in \mathbb{Z}_t$ and invokes the simulator for the proof pointAddition. Zero-knowledge follows from the zero-knowledge of the proof π.

- **Soundness.** We show an extractor that computes λ, x, y given two accepting transcripts for two different challenge bits. Say, we have two accepting views for challenge bits c and $\hat{c} \neq c$. We have,

  $$\lambda = z_1 - \hat{z}_1 \mod t$$

  From the soundness of proofs π, we can extract x, y, γ_1, γ_2 such that $L_x(x, y, \gamma_1, \gamma_2) = R_x(x, y, \gamma_1, \gamma_2)$, and $L_y(x, y, \gamma_1, \gamma_2) = R_y(x, y, \gamma_1, \gamma_2)$. Thus, $T = (\gamma_1, \gamma_2) - (x, y) = \hat{z}_1P - \lambda P$. Thus $\lambda P = (x, y)$.

□

4.5 Proof of Solvency

In this section, we show how to use our constructions for proving composite statements in zero-knowledge to build a privacy-preserving proof of solvency for Bitcoin exchanges. A proof of solvency demonstrates that an exchange controls sufficient reserves to settle each customer's account. If the exchange loses a large amount of money in an attack, it would not be able to provide such a proof. Thus customers will find out about the attack very soon and take necessary actions. We use our SNARK on committed input and output constructions from Section 3.2, and the double discrete logarithm proof in elliptic curve groups from Figure 4.7, to prove a combination statement that is necessary for a proof of solvency for Bitcoin.

4.5.1 Proof of assets

We give the proof of assets in Figure 4.8, which allows an exchange to generate a commitment to its total assets along with a zero-knowledge proof that the exchange knows the private keys for a set of Bitcoin addresses whose total value is equal to the committed value. g is a fixed public generator of a group of order q. For a bitcoin public key y, $x \in \mathbb{Z}_q$ is the corresponding secret key such that $y = g^x$.

We assume that the bitcoin addresses available are the hashes of the public keys. $h = H(y)$ is the bitcoin address corresponding to the key y. We denote the balance associated with an address h by $bal(h)$. The exchange creates a set of public keys $\mathcal{PK}$ to serve as an anonymity set.

$$\mathcal{PK} = \{y_1, \cdots, y_n\} \subseteq G$$

Let $x_1, \cdots, x_n$ be the corresponding secret keys, so that $h_i = H(g^{x_i})$. $s_i \in \{0, 1\}$ indicates whether the corresponding public key in the set is controlled by the exchange. The total assets can now be expressed as:

$$\mathsf{Assets} = \sum_{i=1}^{n} s_i \cdot bal(h_i)$$

The public data available from the blockchain: $h_i = H(y_i), p_i = g^{bal(h_i)}$.

4.5.2 Proof of liabilities

The proof of liabilities given in Figure 4.9, has the exchange commit to its total liability, and in addition, convince all its customers of the inclusion of their balances in the commitment. Consider the mapping of real customers to entries on a liability list. Each real customer should have an entry in the list and no distinct customers is given the same entry. To ensure an injective mapping, customers are provided with an identifier, and in step (d), each customer commits to the unique information $user_i$ (which could potentially include username, email address, or account number). The commitment is binding, preventing the exchange from opening a CID to distinct data for different users. It is also hiding, preventing an adversary who knows the $user_i$ of a potential customer from determining if that customer is in the Liability List. Since we only need hiding and binding and not additive homomorphism, we use a hash-based commitment scheme. We do not require the mapping to be surjective: The exchange can always add fake users to the list, but we need to ensure that doing so can only increase the exchange's apparent liabilities. By including fake users with a zero (or tiny) balance, the exchange can obscure the total number of customers it truly has. However, we need to ensure that any included users can only add to the exchanges total liabilities. That is, the exchange should not be able include a negative balance to try to decrease its apparent liabilities. The requirement, therefore, is that when added together, the sum will never exceed the order of the group. This is enforced in the protocol by having the exchange give a proof that each committed balance is in an interval between 0 and $\mathsf{Max} = 2^{51}$. While the Provisions, the proof of solvency proposed in [DBB+15] achieves this range proof by using bitwise commitments (which contributes to the bulk of the proof size), our comInSnark protocol for zk-SNARK on committed input allows us to use a circuit to check the range instead.

Figure 4.8: Proof of assets

- Commitments: For $i \in [1, n]$, commit to x_i and y_i by publishing $\alpha_i = g^{x_i} h^{a_i}$, $\beta_i = g^{y_i} h^{b_i}$.
- The prover commits to the balance in each address for the public keys he controls and to 0 otherwise, by publishing $u_i = g^{s_i \cdot bal(h_i)} h^{r_i} = p_i^{s_i} h^{r_i}$, $s_i \in \{0, 1\}$, $s_i = 1$ when the prover knows x_i such that $y_i = g^{x_i}$.
- Let f_1 be defined as follows:

$$f_1(y, h) = \begin{cases} 1 & \text{if } H(y) = h \\ 0 & \text{otherwise} \end{cases}$$

  Let f_2 be defined as:

$$f_2(x, y) = \begin{cases} 1 & \text{if } y = g^x \\ 0 & \text{otherwise} \end{cases}$$

  The prover uses the protocol ec-ddlog for f_2, SNARK on committed input and output comIOSnark for f_1 and the OR composition to prove the following, for each i:
  π_i : $\mathsf{PK}\{(y_i, x_i, s_i, r_i, a_i, b_i)$: $(\alpha_i = g^{x_i} h^{a_i} \wedge \beta_i = g^{y_i} h^{b_i} \wedge f_2(x_i, y_i) = s_i \wedge f_1(y_i, h_i) = s_i \wedge u_i = g^{s_i \cdot bal(h_i)} h^{r_i} \wedge s_i = 1) \vee (s_i = 0)\}$
- Compute and publish $Z_{Assets} = \prod_{i=1}^{n} u_i$

Figure 4.9: Proof of liabilities

(a) Let C be a circuit that takes as input m bit integers $x_1, \cdots, x_n$ and outputs 1 if $x_i < \mathsf{Max}$ for all i and 0 otherwise.

(b) The prover commits to each customer $\mathcal{C}_i$'s balance x_i by publishing $c_i = g^{x_i} h^{r_i}$

(c) The prover gives a SNARK on committed input that $x_i < Max$ for all customers. The prover uses $\mathsf{comInSnark}$ to give a ZK proof π that $C(x_1, \cdots, x_n) = 1$ given c_i. $\mathsf{PK}\{(x_i, r_i) : C(x_1, \cdots, x_n) = 1 \wedge c_i = g^{x_i} h^{r_i}\}$.

(d) The prover computes a customer identifier for each customer by choosing a random nonce and computing

$$\mathsf{CID}_i = H(user_i || n_i)$$

where $n_i \in \{0,1\}^{512}$, $user_i$ is the ith customer's username, and H is a collision resistant hash function.

(e) Publish the liabilities list of all customers' tuples.

$$\mathsf{ListLiab} = (\mathsf{CID}_1, \cdots, \mathsf{CID}_n, c_1, \cdots, c_n, \pi)$$

(f) Each client is privately given (r_i, n_i)

  (a) The client computes CID and verifies inclusion in the liabilities list.

  (b) The client checks its own balance is included by computing $c_i = g^{balance_i} h^{r_i}$

  (c) Verifies the proof π

  (d) Each client computes $Z_{Liab} = \prod_{i=1}^{n} c_i$

Figure 4.10: Proof of solvency

1. The exchange uses the proof of assets in Figure 4.8 and generates a commitment to its total assets Z_{Assets}.
2. The exchange uses the proof of liabilities in Figure 4.9 to generate a commitment to its total liabilities Z_{Liab} and a list of its liabilities, ListLiab.
3. The exchange gives a proof $\pi : \mathsf{PK}\{(R) : Z = h^R\}$, where $Z = Z_{Assets} \cdot Z_{Liab}^{-1}$.

4.5.3 Privacy-preserving proof of solvency

We assume that each customer checks the liability list published by the exchange as part of the proof of liabilities to verify the inclusion of their customer CID and the correctness of committed balance. Thus, in step (f), the customers need to perform step (f)b. Steps (f)c and (f)d, on the other hand maybe performed by an auditor on behalf of the customers. Given the proofs in Figures 4.8 and 4.9, the proof of solvency involves the exchange proving that $Z_{Assets-Liab}$ is a commitment to 0. The proof of solvency is given in Figure 4.10.

As suggested in Provisions, the case where the exchange is actually running a surplus, and the total assets are greater than total liabilities, can be handled with a simple modification: the exchange can create a commitment to its surplus, and apply the same range proof used for customer balances to prove that this is a small positive number. It then replaces the final step in Figure 4.10 to $Z = Z_{Assets} \cdot Z_{Liab}^{-1} \cdot Z_{surplus}^{-1}$. The exchange could also move its surplus into a separate address and not include it in the addresses used in its proof of assets, or include the value of the surplus in a number of fake customers' accounts if it is desirable to hide even the existence of any surplus.

Chapter 5

Hashing Garbled Circuits for Free [1]

In this chapter, we present a definition for hash security of garbled circuits, present constructions that satisfy our definition and show applications to zero-knowledge and general secure two-party computation. We begin this chapter with a high-level technical overview of our approach, then present our Free Hash construction.

5.1 Overview

We take advantage of the observation that the input to the hash is a garbled circuit GC, which must be evaluatable using the garbled circuit Eval function. We will not require standard hash collision resilience of GC strings, achieving which is very costly relative to the cost of GC generation. Instead, we guarantee that if an adversary can find another string $\widehat{\mathsf{GC}}$ that matches the hash of a correctly garbled GC, then with high probability, the garbled circuit property of $\widehat{\mathsf{GC}}$ is broken and its evaluation will fail.

We present our intuition iteratively; we start with a naive efficient approach, which we then refine and arrive at a secure hashed garbling. Recall, we start with a correctly generated GC GC with the set of output decoding labels d. The adversary's goal is to generate a circuit $\widehat{\mathsf{GC}}$ with the same hash as GC, and which will not fail evaluation/decoding given *the same* output labels d. This hash guarantee is sufficient for certain GC-based SFE protocols. A syntactic difference with [GMS08] C&C hashing is that verification of Free Hash involves GC evaluation, and is only

[1] This chapter is primarily based on joint work with Xiong Fan and Vladimir Kolesnikov, that appeared in Eurocrypt 2017 [FGK17]. Certain passages have been taken from this source verbatim.

possible once input labels are received (e.g., after OT of input labels). More importantly, Free Hash, as applied to C&C, provides a security guarantee subtly distinct from collision-resistance. Hence, drop-in replacement of [GMS08] C&C hashing with Free Hash may not be always possible, and in general should be done by hand and original proofs re-checked. See Section 5.2.4 for additional discussion.

We present the intuition for the classical four-row GC; we use similar ideas to achieve half-gates GC hashing as well. We present and prove secure both Free Hash constructions.

The first Free Hash idea is to simply set the hash of the garbled circuit to be the XOR of all garbled table (GT) rows of GC. This is clearly problematic, since a cheating garbler $\mathcal{A}$ can mount, for example, the following attack. $\mathcal{A}$ will set one GT entry to be the encryption of the wrong wire label. This affects the XOR hash as follows $H(\widehat{\mathsf{GC}}) = H(\mathsf{GC}) \oplus \Delta$. Now suppose the garbler knows (or guesses) which GT entry anywhere in GC will not be used in evaluation (inactive GT row). Now $\mathcal{A}$ simply replaces the inactive GT row X with value $X \oplus \Delta$. This will restore the hash to the desired value, and since this entry will not be used in the evaluation, the garbler will not be caught.

The following refinement of this approach counters the above attack: we make the gate's output wire key depend (in an efficient manner) on *all* GT rows of that gate. The idea is that XOR hash correction, such as above, will necessarily involve modification to an active GT row, which will affect the computed wire key on that gate. Importantly, because wire keys and GT rows are related via a random (albeit known) function, a GT row offset by Δ (needed to "fix" the hash) will result in effectively randomizing the output wire label of the gate. Because a non-failing evaluation requires output wire labels to be consistent with the fixed decoding information d, $\mathcal{A}$ will now be stuck.

We attempt this by starting with a secure garbling scheme $\mathcal{G}$, and modifying the way the wire labels are defined, as follows. The two wire labels w_i^0, w_i^1 associated with gate G_i's output wire will now be treated as temporary labels. A label W_i^j of the new scheme will be obtained from the w_i^j simply by XORing it with all the GT rows of G_i.

This is not quite sufficient, as it still allows the attacker to modify a GT row and then correct it within *the same* gate table. This is possible since a "fix" for the hash does not disrupt the validity of the wire label, as both the hash and the new wire label are defined in the same manner (as XOR of all the GT rows of G_i). Our final idea, is to use the GT rows as XOR pads in a different manner for computing the GC hash and for offsetting the wire values. This way, the fix for the hash w.h.p. will not simultaneously keep the wire label valid. We achieve this by malleating GT rows prior to using them as XOR pads in wire value computation.

It is not hard to show that the above changes preserve the privacy and authenticity properties of the garbling scheme.

We summarize the intuition for the hash security of the above construction. Consider a $\widehat{\mathsf{GC}} \neq \mathsf{GC}$ that collides under the above hash. Then, the evaluation of $\widehat{\mathsf{GC}}$ will deviate from that of GC w.r.t. some wire label. Importantly, $\widehat{\mathsf{GC}}$ evaluation can subsequently either return to a valid wire label or to a correct running hash, but not both. Thus, evaluation of $\widehat{\mathsf{GC}}$ using encoding information $\widehat{e}$ cannot go back to both the wire label and the hash being correct.

A formalization of what precisely the GC description string GC includes is often natural and hence is usually omitted from discussion. In our setting this an important aspect, as we focus on the collision resilience-related properties of GC strings, as well as on minimizing the size of GC and its computation time.

Firstly, we remind the reader that in the BHR [BHR12] notation the function Gb outputs the garbling *function* F. Since it is problematic to operate on functions, BHR regards Gb as operating on strings representing and defining the corresponding functions. In our notation, Gb outputs GC, which we treat as a string defining the evaluation process as well.

Clearly, GC will contain a set of garbled tables; the question is how to treat the circuit topology, i.e. exactly how to describe/define how Eval should process GC. One choice is to treat the plaintext circuit/topology as a part of GC. Because we focus on size/computation, this approach would cause some waste. Indeed, in most scenarios, the circuit and topology is known to both players, and hence could be implicit in GC.

Instead, we opt to consider the circuit description, including the locations of the free XOR gates as an externally generated string. This is certainly the case in SFE where the evaluated function is known to both players, and players can *a priori* adopt a convention on how to map the GC garbled gates to the circuit gates, hence defining the evaluation process. In Private Function Evaluation (PFE), which is the case in our certified function evaluation scenario (see Section 1.3.2), the evaluated function is *not* known to the evaluator. In this case, we still treat the topology/evaluation instructions as external to GC and assume that they are correctly delivered to the evaluator. We note that in the certified function case, this can be naturally achieved by the CA signing the topology with a unique identifier, and including this identifier with GC and the hash of GC.

Our Assumptions. Our work optimizes high-performance primitives, and it is important to be clear on the assumptions we require of them so as to properly compare to related work.

We use the same primitives, and nearly identical constructions as JustGarble [BHKR13] and half-gates [ZRE15]. As a result, privacy and authenticity properties of our schemes hold under the same assumptions as [BHKR13, ZRE15], namely that the key derivation function used in garbling is a Davies-Meyer (DM) construction in the random-permutation model (RPM). While [BHKR13] proves

the security of their construction in the RPM directly, [ZRE15] abstracts the DM security property as a variant of correlation-robust function. To achieve hash security, we need to assume collision resistance of DM. We give the definition of a collision resistant hash function below.

Collision-resistant hash function. A hash function family $\mathcal{H}$ is a collection of functions, where each $H \in \mathcal{H}$ is a mapping from $\{0,1\}^m$ to $\{0,1\}^n$, such that $m > n$ and m, n are polynomials in security parameter κ. An instance $H \in \mathcal{H}$ can be described by a key which is publicly known. We say a hash function family $\mathcal{H}$ is *collision-resistant* if for any PPT adversary $\mathcal{A}$

$$\Pr\left(x \neq x' \wedge H(x) = H(x') \,:\, (x, x') \leftarrow \mathcal{A}(H), H \xleftarrow{R} \mathcal{H}\right) = \mathsf{negl}(\kappa)$$

Cipher Instantiation. We instantiate the key derivation function (KDF) calls as do [BHKR13, ZRE15], with the Davies-Meyer construction. Namely, the input X to KDF $H(X, i)$ are the 128-bit long wire keys, and i is an internal integer that simply increments per hash function call. We set $H_\pi(X, i) = \pi(K) \oplus K$, where $K = 2X \oplus i$ (π is assumed to be an ideal cipher, instantiated with 128-bit AES with randomly chosen key).

Ideal cipher model. The Ideal Cipher Model (ICM) is an idealized model of computation, similar to the random oracle model (ROM) [BR93b]. In the ICM, one has a publicly accessible random block cipher (or ideal cipher). This is a block cipher with a k-bit key and a n-bit input/output, that is chosen uniformly at random among all block ciphers of this form; this is equivalent to having a family of 2^k independent random permutations. All parties including the adversary can make both encryption and decryption queries to the ideal block cipher, for any given key. The ICM is shown to be equivalent to ROM [CPS08, HKT11].

Hash Security Parameters. We use κ = 128-bit security parameter, which is standard for encryption and GCs. However, 128-bit hash domain is often seen as insufficient. This is because of the birthday attack, which provides time-space tradeoff for an attacker. Specifically, a collision-finding attacker can precompute and store a square-root number of hash images. Then by birthday paradox, a random collision will be found among these images with significant probability. This attack requires 2^{64} hash computations and efficiently accessible storage for 2^{64} hash values. We argue that 128-bit hash security is nevertheless acceptable in SFE, if used carefully. Importantly, hash checks in two party computation have an *online* property, meaning that we can set up the system such that preprocessing or post-processing will not aid the attacker. Indeed, consider the SFE scenario

and the following solution. In the existing fixed-key cipher-based protocols it is specified that the fixed key is chosen at random prior to GC generation. We can simply explicitly require that *both* players contribute to key generation, and that the selected key will be the one defining the fixed-key permutation used in GC. This will render any precomputation useless. Post-computation, while a threat to the privacy and, perhaps, authenticity of GC, is not helping the attacker, since the GC evaluator decision to accept or reject reached during the execution, is irrevocable. GC evaluator can set a generous time limit (e.g. several seconds or even minutes) after which it will abort the execution.

5.1.1 Related Work

To our knowledge, there is no prior work specifically addressing hashing of GCs. At the same time, significant research effort has been expended on optimizing core GC performance. Work includes algorithmic GC improvements, such as Free XOR [KS08a], FleXOR [KMR14], half-gates [ZRE15], as well as optimizing underlying primitives, such as JustGarble [BHKR13]. Our work complements the existing GC improvement work.

Of course, the natural GC hashing approach works: just hash the generated GC. The problem with this is, of course, its cost. Relative cost of fixed-key cipher garbling and hashing are strongly architecture-dependent. They can be almost the same (e.g., when both AES and SHA are implemented in hardware). In another extreme, Intel's white paper [GGO+] reports that AES-NI evaluation of 16-byte blocks is 23× faster that that of SHA1 (35,965.9 vs 793,718.7 KB/sec). In our experiments reported in Section 5.2.4, we observed about 6× performance difference between AES-NI and SHA1.

Improving on this, and motivated in part by the availability of fast hardware AES implementations, there was a short series of works [BRS02, RS08b, RS08a, BÖS11], implementing a hash function with three fixed-key AES function calls. A recent work of Rogaway and Steinberger [RS08a] constructs a class of linearly-determined, permutation-based compression functions $\{0,1\}^{mn} \to \{0,1\}^{rn}$ making k calls to the different permutations π_i for $i \in [k]$, where they named their construction as LPmkr. The fastest construction LP362 (12.09 cycles per byte) [BÖS11], with 6 calls to fixed-key AES would cost about 6× of that of fast garbling. Davies-Meyer-based hash construction [Win84] in the ideal cipher model considered in literature is reported to have similar speeds [BÖS11].

In comparison, our work eliminates the cost of hash whatsoever, while adding no cost to garbling or GC evaluation.

C&C and uses of hashed GC. There is a long sequence of GC-based SFE work, e.g. [Lin13, HKE13, Bra13, LR14, HKK+14, KM15], most of which uses

some form of C&C or challenging the GC generator. Based on [GMS08], these works will benefit from our result, to varying degree. The exact performance benefit will depend on where the Free Hash is used, the ratio of evaluated/test circuits, as well as the computational/communication resources available to the players. In Section 5.2.4, we calculate performance improvement in several C&C protocols due to our GC hash, and in Section 5.3, we discuss the application of Free Hash to zero-knowledge.

5.2 GC hashing scheme

In this section, we define our hashed garbled circuit scheme. We capture the security guarantees we require from this new notion, and then present our construction that outputs a garbled circuit and its hash. Our garbled circuit construction satisfies the properties of correctness, authenticity and privacy. We then show that our construction is secure according to our hash security definition.

5.2.1 Hashed Garbled Circuit security

Recall, we want to define hash security of garbled circuits with the same topology. We require that if the hash of such two garbled circuits collide, and one of them verifies correctly, then with high probability the other garbled circuit will fail evaluation. We now formalize this intuition in the definition below.

Definition 5.2.1. *(Hash security) A garbling scheme $\mathcal{G}$ is* hash-secure *with respect to a hash function $\mathcal{H}$ if for every boolean circuit $\mathcal{C}$, input x and* PPT *adversary $\mathcal{A}$,*

$$\Pr\left(\begin{array}{c} \mathsf{De}(\mathsf{Eval}(\widehat{\mathsf{GC}},\mathsf{En}(x,\widehat{e})),d)\neq\perp \\ \mathsf{GC}\neq\widehat{\mathsf{GC}}, \\ \mathsf{Topology}(\mathsf{GC})=\mathsf{Topology}(\widehat{\mathsf{GC}}), \\ \mathsf{Ve}(\mathcal{C},\mathsf{GC},d,e)=\mathsf{accept}, \\ \mathcal{H}(\mathsf{GC})=\mathcal{H}(\widehat{\mathsf{GC}})=h \end{array} : \begin{array}{c} \Big(\mathsf{GC},\widehat{\mathsf{GC}},e=\{X_j^0,X_j^1\}_{j\in[m]}, \\ \widehat{e}=\{\widehat{X}_j^0,\widehat{X}_j^1\}_{j\in[m]},d,h\Big)\leftarrow\mathcal{A}(\mathcal{C},1^\kappa), \end{array}\right)$$

is negligible in κ.

We point out that the decoding information d that results in failed decoding of $\widehat{\mathsf{GC}}$ is the same decoding information with respect to which GC successfully verifies, and this is essential to hash security. If we did not place this requirement, then an adversary can change d to $\widehat{d}$ which decodes any string that Eval on $\widehat{\mathsf{GC}}$ returns. We note that in full generality it is not necessary to require $\mathcal{A}$ to generate a GC passing the verification Ve of a specific circuit $\mathcal{C}$. We can achieve that if an $\mathcal{A}$ generates two unequal GCs with the same hash, at least one of them will always

output $\perp$. However, the above definition 5.2.1 reflects the typical use of GCs, and is sufficient for our construction.

In this work, we consider verifiable garbling schemes with hash security. That is, $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Eval}, \mathsf{De}, \mathsf{Ve}, \mathcal{H})$. Because we apply our constructions to secure computation, we will need schemes additionally satisfying the properties of correctness (cf. Definition 2.3.1) and privacy (cf. Definition 2.3.2). If needed, the authenticity property of GC (cf. Definition 2.3.3) can be achieved as well.

5.2.2 Our Construction

We now formalize the intuition of Section 5.1 on how to generate a GC hash for free when garbling. The full construction is presented in Figure 5.1; here we provide additional intuition. Recall, in Section 5.1, we explained that after we generated (temporary) GC tables, we need to XOR their GT entries into the GC hash in one manner, and into the GC wire labels in another manner. In our construction, we do so by bitwise shifting the GT entries C_i prior to XORing them into the wire labels.

We note that we use bit shifting because it is fast and easy to implement, but a more general condition is sufficient for security of our scheme, which is as follows: We set the wire labels of a gate output wire as a function of its temporary wire labels and the entries of the garbled gate table. Consider functions f_i such that, if

$$\bigoplus_{i=1}^{4} C_i = \bigoplus_{i=1}^{4} \widehat{C}_i$$

for some $C_i \neq \widehat{C}_i$, then,

$$\Pr[\bigoplus_{i=1}^{4} f_i(C_i) = \bigoplus_{i=1}^{4} f_i(\widehat{C}_i)]$$

is negligible. As we will later see in the proof, this is the property that we use in proving the hash security of our construction in proof of Theorem 5.2.4.

In presenting our construction, we adopt the approach used by [BHKR13] and others, where the gates are garbled as $H(w_i||w_j||r) \oplus w_k$, where w_i and w_j are wire labels on input wires, r is a nonce and w_k is a wire label on the output wire. H is a key-derivation function modeled as a random oracle.

The scheme we present below follows the standard point-and-permute optimization. This was introduced by Beaver, Micali and Rogaway in [BMR90], where a select bit is appended to each wire label, such that the two labels on each wire have opposite select bits. This association between select bits and the logical truth values is random and kept secret. Now the garbled truth table can be arranged by

these public select bits. The evaluator can select the correct ciphertext to decrypt based on the select bit instead of trying all four. For each wire label w, its least significant bit $\mathsf{lsb}(w)$ is reserved as a select bit that is used as in the point-and-permute technique, and complementary wire labels have opposite select bits. For the ith wire, define $p_i = \mathsf{lsb}(w_i^0)$. When using Free XOR, the global randomly chosen offset R is such that $\mathsf{lsb}(R) = 1$. Since $w_i^0 \oplus w_i^1 = R$ holds for each i in the circuit, we have that $\mathsf{lsb}(w_i^0) \neq \mathsf{lsb}(w_i^1)$.

To simplify presentation, in our constructions and notation we set the decoding information simply to be the output wire labels. We note, this does not preserve the authenticity property of GC. Authenticity can be easily achieved in our scheme, e.g. by instead setting the decoding information to be the collision-resistant hashes of the output labels. In more detail, let H be a collision-resistant hash function. The output translation table for a wire will now be $\{H(w_i^0), H(w_i^1)\}$. Given a garbled value w_i^b on an output wire, it is possible to determine whether it corresponds to the 0 or 1 key by computing $H(w_i^b)$ and checking whether it is equal to the first or second value in the pair. However, given this output translation table, it is not feasible to find the actual garbled values.

Let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a function, satisfying properties discussed in Section 5.1. For a function represented by a circuit $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$, we use $W_{\mathsf{in}}, W_{\mathsf{out}}$ to denote the input and output wires of f respectively, and G_{inter} for intermediate gates. The Free Hash garbling scheme $\mathsf{h}\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Eval}, \mathsf{Ve}, \mathcal{H})$ is described in Figure 5.1.

- $\mathsf{Gb}(1^\kappa, \mathcal{C})$: On input the security parameter κ and a circuit $\mathcal{C}$, choose $R \leftarrow \{0,1\}^{\lambda-1}||1$ and set $h = 0$.
  1. For each input wire $W_i \in W_{\mathsf{in}}$ of the circuit $\mathcal{C}$, set garbled labels in the following way: Randomly choose $K_i^0 \in \{0,1\}^\kappa$. Set $K_i^1 = K_i^0 \oplus R$. Set the garbled labels for input wire W_i as $w_i = (K_i^0, K_i^1)$.
  2. For each intermediate gate $G_i : W_c = g_i(W_a, W_b)$ of $\mathcal{C}$ in topological order:
     (a) Parse the garbled input labels as $w_a = (K_a^0, K_a^1)$ and $w_b = (K_b^0, K_b^1)$.
     (b) If G_i is an XOR gate, set garbled labels for the gate output wire W_c as $K_c^0 = K_a^0 \oplus K_b^0$, and $K_c^1 = K_c^0 \oplus R$.
     (c) If G_i is an AND gate
        – Choose temporary garbled labels for the gate output wire W_c as $T_c^0 \in \{0,1\}^\kappa$, and set $T_c^1 = T_c^0 \oplus R$.
        – Create G_i's garbled table: For each possible combination of G_i's input values $v_a, v_b \in \{0,1\}$, set $\tau^i_{v_a,v_b} = H(K_a^{v_a}|K_b^{v_b}|i) \oplus T_c^{g_i(v_a,v_b)}$. Sort entries τ^i in the table by input pointers, and let the entries be $C_{i,1}, C_{i,2}, C_{i,3}, C_{i,4}$.

– For $d \in \{0,1\}$, compute:

$$\mathsf{pad}_{i,1} = C_{i,1}^{\ll 1} \oplus C_{i,2}^{\ll 2} \oplus C_{i,3}^{\ll 3} \oplus C_{i,4}^{\ll 4}$$

$$K_c^0 = T_c^0 \oplus \mathsf{pad}_{i,1}$$

Set the garbled labels for wire W_c as

$$w_c = (K_c^0, K_c^1), \text{ where } K_c^1 = K_c^0 \oplus R$$

– Define

$$\mathsf{pad}_{i,2} = C_{i,1} \oplus C_{i,2} \oplus C_{i,3} \oplus C_{i,4}$$

$$h = h \oplus \mathsf{pad}_{i,2}$$

3. For each output wire $W_i \in W_{\mathsf{out}}$ of $\mathcal{C}$, set $d_i^0 = (0, K_i^0)$ and $d_i^1 = (1, K_i^1)$
4. Output encoding information e, decoding information d, garbled circuit GC and hash $\mathcal{H}(\mathsf{GC})$ as

$$e = \{(K_i^0, K_i^1)\}_{W_i \in W_{\mathsf{in}}}, d = \{(d_i^0, d_i^1)\}_{W_i \in W_{\mathsf{out}}}, \mathsf{GC} = \{\tau_{a,b}^i\}_{\substack{a,b \in \{0,1\} \\ G_i \in G_{\mathsf{inter}}}}, \mathcal{H}(\mathsf{GC}) = h$$

- $\mathsf{En}(\boldsymbol{x}, e)$: On input encoding information e and input $\boldsymbol{x}$, output encoding $\mathbf{X} = \{X_i^{\boldsymbol{x}[i]}\}_{i \in [n]}$.
- $\mathsf{De}(\mathbf{Y}, d)$: On input the decoding information d and the garbled output of the circuit $\mathbf{Y} = (Y_1, ..., Y_m)$, for each output wire i of the circuit $\mathcal{C}$, parse d as $d = \{(0, K_i^0), (1, K_i^1)\}_{i \in [m]}$. Then, set $y_i = b$ if $Y_i = K_i^b$ and $y_i = \perp$ if $Y_i \notin \{K_i^0, K_i^1\}$. Output the result $\boldsymbol{y} = (y_1, ..., y_m)$ if $\forall i, y_i \neq \perp$. Else, output $\perp$.
- $\mathsf{Eval}(\mathsf{GC}, \mathbf{X})$: On input the garbled circuit GC and garbled input $\mathbf{X}$, for each gate $G_i : W_c = g_i(W_a, W_b)$ with garbled inputs $w_a = K_a^{v_a}, w_b = K_b^{v_b}$. If G_i is an XOR gate, compute $w_c^{g_i(v_a, v_b)} = K_a^{v_a} \oplus K_b^{v_b}$. If G_i is an AND gate:
  1. Let C_1, C_2, C_3, C_4 be the table entries. Compute $\mathsf{pad} = \bigoplus_{i=1}^{4} C_i^{\ll i}$.
  2. Decode the temporary output value from garbled table entry τ^i in position (v_a, v_b) as $T_c^{g_i(v_a, v_b)} = H(K_a^{v_a} | K_b^{v_b} | i) \oplus \tau^i$.
  3. Compute the garbled value as $w_c^{g_i(v_a, v_b)} = T_c^{g_i(v_a, v_b)} \oplus \mathsf{pad}$.
- $\mathsf{Ve}(\mathcal{C}, \mathsf{GC}, d, e)$: Check that each gate in GC correctly encrypts the gate in $\mathcal{C}$ given the encoding information e. If yes, then output accept, else output reject.
- $\mathcal{H}(GC)$: On input the garbled circuit GC, output h as the XOR of all ciphertexts,

$$h = \bigoplus_{g_i} (C_{i,1} \oplus C_{i,2} \oplus C_{i,3} \oplus C_{i,4})$$

Figure 5.1: The Free Hash garbling scheme $\mathsf{h}\mathcal{G}$

The construction in Figure 5.1 satisfies the properties of authenticity (cf. Definition 2.3.3), privacy (cf. Definition 2.3.2) and hash security (cf. Definition 5.2.1). Our changes to the standard construction do not affect the other GC properties. The proofs of privacy and authenticity closely follow the arguments of [BHKR13] and [ZRE15], and use the weakened definition of circular correlation robustness of a function H, in [ZRE15] (Definition 3 of [ZRE15]). We prove the hash security of our construction below.

Hash security. We now state and prove a technical lemma on which we rely for proving hash security (Theorem 5.2.4). The lemma below captures the following useful fact about GC and $\widehat{\mathsf{GC}}$: a gate in $\widehat{\mathsf{GC}}$ whose $\mathsf{pad}_{i,2}$ (XOR hash of the gate table) collides with that of the gate in GC will not be evaluated correctly (i.e. will not produce a valid label on the output wire) if the gate table is different, or if the input wire keys of the gate are different, or both. We say that a wire label, obtained during evaluation on input x encoded using $\widehat{e}$, is valid if it is one of the two possible wire labels for the same wire in GC. For presentation, we slightly abuse notation, by writing g_i to mean both the gate and the garbled table corresponding to the gate. It will be clear from context, which of the two is meant.

Definition 5.2.2. *(Valid key) Let* $(\mathsf{GC}, e, \widehat{\mathsf{GC}}, \widehat{e}, d, h)$ *be such that* $\mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathsf{Topology}(\mathsf{GC}) = \mathsf{Topology}(\widehat{\mathsf{GC}}), \mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}) = h$ *and* $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$. *An internal wire key* $\widehat{K}_i^b$ *obtained on wire* w_i *during* Eval *of* $\widehat{\mathsf{GC}}$ *is called valid if* $\widehat{K}_i^b \in \{K_i^0, K_i^1\}$ *where* (K_i^0, K_i^1) *are the wire keys corresponding to* 0 *and* 1 *on wire* w_i *in* GC.

Lemma 5.2.3. *Let* $(\mathsf{GC}, e, \widehat{\mathsf{GC}}, \widehat{e}, d, h) \leftarrow \mathcal{A}(1^\kappa)$ *be such that* $\mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathsf{Topology}(\mathsf{GC}) = \mathsf{Topology}(\widehat{\mathsf{GC}}), \mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}) = h$ *and* $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$. *Assuming* $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$, *evaluation of the garbled gate* $\widehat{g}_i$ *during* Eval *results in a valid wire label for the output wire of the gate with probability* $\mathsf{negl}(\kappa)$ *in the following cases:*

1. *Input wire keys to gate* $\widehat{g}_i$ *are valid, and* $\widehat{g}_i \neq g_i$.
2. *At least one input wire key to gate* $\widehat{g}_i$ *is invalid and* $\widehat{g}_i = g_i$.
3. *At least one input wire key to gate* $\widehat{g}_i$ *is invalid, and* $\widehat{g}_i \neq g_i$.

Proof. Let $g_i = \{C_1, C_2, C_3, C_4\}$ be the ith garbled table in GC and $\widehat{g}_i = \{\widehat{C_1}, \widehat{C_2}, \widehat{C_3}, \widehat{C_4}\}$ the ith garbled table in $\widehat{\mathsf{GC}}$.

Case 1 Since $\widehat{g}_i \neq g_i$, w.l.o.g., let $C_1 \neq \widehat{C_1}$. Since $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$, there must be (at least) one $j \neq 1$ such that $\widehat{C_j} \neq C_j$. Now, $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$ gives,

$$\widehat{C_j} \oplus \widehat{C_1} = C_j \oplus C_1 \tag{5.1}$$

Let $\widehat{K} = (\widehat{K}_a, \widehat{K}_b)$ be the input wire key to gate g_i in $\widehat{\mathsf{GC}}$ during Eval, which by assumption is valid.

For the sake of contradiction, say, one of the ciphertexts, say, $\widehat{C_1}$, in $\widehat{g_i}$ gives a valid output wire key. Let T be the intermediate key obtained by decrypting $\widehat{C_1}$. Now validity of output wire key implies $T \oplus \widehat{\mathsf{pad}}_{i,1} = K \in \{K^0, K^1\}$.

$$T \oplus \widehat{\mathsf{pad}}_{i,1} = K$$
$$\widehat{C_j}^{\ll j} \oplus \widehat{C_1}^{\ll 1} = C_j^{\ll j} \oplus C_1^{\ll 1} \oplus R \tag{5.2}$$

where $R = T \oplus K \oplus \mathsf{pad}_{i,1}$ is a fixed value, and $T = H(\widehat{K}||i) \oplus \widehat{C_1}$. Therefore, K is valid only when both (5.1) and (5.2) hold. We now argue that this happens with probability $\leq 1/2^\lambda$. By the assumption that $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$, C_1 and C_j are random keys masked by the outputs of the function H. If, therefore, a $\widehat{C_1}$ and $\widehat{C_j}$ that satisfies (5.1), also satisfies (5.2), then we can find r_1 and r_2 such that $r_1 \oplus r_2$ is δ for some fixed δ and $r_1^{\ll} \oplus r_2^{\ll}$ collides with the output of the function H on a fixed value. By collision resistance of the function H, this happens only with probability $\leq 1/2^\lambda$.

Case 2 $g_i = \widehat{g_i}$. Either $\widehat{K}_a \notin \{K_a^0, K_a^1\}$ or $\widehat{K}_b \notin \{K_b^0, K_b^1\}$ or both, where (K_a^0, K_a^1) and (K_b^0, K_b^1) are the wire keys corresponding to the input wires of $\widehat{g_i}$ in GC. Let (K^0, K^1) be the wire keys of the output wire of g_i.

For the sake of contradiction, say, one of the ciphertexts, say, C_1, gives a valid output wire key with $\widehat{K}$ as the input wire keys. Let T be the intermediate key obtained by decrypting C_1. Now validity of output wire key implies $T \oplus \mathsf{pad}_{i,1} = K \in \{K^0, K^1\}$. That is,

$$H(\widehat{K}||i) \oplus C_1 \oplus \mathsf{pad}_{i,1} = K \tag{5.3}$$

K is valid when (5.3) holds, and that happens with negligible probability since we can find a r such that the output of H on r collides with a given value only with probability $\leq 1/2^\lambda$.

Case 3 W.l.o.g., let $C_1 \neq \widehat{C_1}$. Since $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$, there must be (at least) one $j \neq 1$ such that $\widehat{C_j} \neq C_j$.

Either $\widehat{K}_a \notin \{K_a^0, K_a^1\}$ or $\widehat{K}_b \notin \{K_b^0, K_b^1\}$ or both, where (K_a^0, K_a^1) and $(K_b^0 K_b^1)$ are the wire keys corresponding to the input wires of $\widehat{g_i}$ in GC. (K^0, K^1) be the wire keys of the output wire of g_i.

Now, $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$ gives,

$$\widehat{C_j} \oplus \widehat{C_1} = C_j \oplus C_1 \tag{5.4}$$

Let $\widehat{K} = (\widehat{K}_a, \widehat{K}_b)$ be the input wire key to gate g_i in $\widehat{\mathsf{GC}}$ during Eval. Since $\widehat{K}$ is invalid by assumption, either $\widehat{K}_a \notin \{K_a^0, K_a^1\}$ or $\widehat{K}_b \notin \{K_b^0, K_b^1\}$ or both, where (K_a^0, K_a^1) and (K_b^0, K_b^1) are the wire keys corresponding to the input wires of $\widehat{g}_i$ in GC. (K^0, K^1) be the wire keys of the output wire of g_i.

For the sake of contradiction, say, one of the ciphertexts, say, $\widehat{C_1}$, in $\widehat{g}_i$ gives a valid output wire key. Let T be the intermediate key obtained by decrypting $\widehat{C_1}$. Now validity of output wire key implies $T \oplus \widehat{\mathsf{pad}}_{i,1} = K \in \{K^0, K^1\}$.

$$\begin{aligned} T \oplus \widehat{\mathsf{pad}}_{i,1} &= K \\ \widehat{C_j}^{\ll j} \oplus \widehat{C_1}^{\ll 1} &= C_j^{\ll j} \oplus C_1^{\ll 1} \oplus R \end{aligned} \tag{5.5}$$

where $R = T \oplus K \oplus \mathsf{pad}_1$, and $T = H(\widehat{K}||i) \oplus \widehat{C_1}$. Therefore, K is valid only when both (5.4) and (5.5) hold. We now argue that this happens with probability $\leq 1/2^\lambda$. By the assumption that $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$, C_1 and C_j are random keys masked by the outputs of the function H. If, therefore, $\widehat{K}$, $\widehat{C_1}$ and $\widehat{C_j}$ satisfy (5.4) and (5.5), then we can find r, r_1 and r_2 such that the output of the function H on r collides with $r_1^{\ll} \oplus r_2^{\ll}$ and $r_1 \oplus r_2$ is δ for some fixed δ. By collision resistance of the function H, this happens with probability at most $1/2^\lambda$.

When there is more than one $j \neq 1$ such that $\widehat{C_j} \neq C_j$ in cases (1) and (3) above, we will have,

$$\begin{aligned} \bigoplus_{j \neq 1} \widehat{C_j} \oplus \widehat{C_1} &= \bigoplus_{j \neq 1} C_j \oplus C_1 \\ \bigoplus_{j \neq 1} \widehat{C_j}^{\ll j} \oplus \widehat{C_1}^{\ll 1} &= \bigoplus_{j \neq 1} C_j^{\ll j} \oplus C_1^{\ll 1} \oplus R \end{aligned}$$

and the same arguments extend. □

Theorem 5.2.4. *The Free Hash garbling scheme* $\mathsf{h}\mathcal{G}$ *described in Figure 5.1 satisfies hash security as defined in Definition 5.2.1 assuming the collision-resistance of* H*.*

Proof. Given an adversary $\mathcal{A}$ who outputs $(\mathsf{GC}, e, \widehat{\mathsf{GC}}, \widehat{e}, d, h)$ such that $\mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}) = h$, $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$, we show that $\forall x, \Pr[\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(x, \widehat{e})) \neq \bot] = \mathsf{negl}(\kappa)$. Since $\mathsf{GC} \neq \widehat{\mathsf{GC}}$, they differ in at least one garbled gate. Let g_i be the first gate in topological order that differs in GC and $\widehat{\mathsf{GC}}$. When $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$ for all $\widehat{g}_i \neq g_i$, by case (1) of Lemma 5.2.3, we have that the output wire key for $\widehat{g}_i$ is invalid. Now, by inductively applying cases (2) and (3) of Lemma 5.2.3, all wire keys from then on, in topological order of evaluation remain invalid.

Now, when $\mathsf{pad}_{i,2} \neq \widehat{\mathsf{pad}}_{i,2}$, Eval on $\widehat{\mathsf{GC}}$ can return to a valid wire key for the output wire of $\widehat{g}_i \neq g_i$. Let us denote by $\widehat{\mathcal{H}}_i$ the running hash up until gate $\widehat{g}_i$ in $\widehat{\mathsf{GC}}$. Since $\mathsf{pad}_{i,2} \neq \widehat{\mathsf{pad}}_{i,2}$, we have $\widehat{\mathcal{H}}_i \neq \mathcal{H}_i$. By the assumption that $\mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC})$, there must be a gate $\widehat{g}_j \neq g_j$ such that

$$\Delta = \bigoplus_{i:\mathsf{pad}_{i,2} \neq \widehat{\mathsf{pad}}_{i,2}} (\mathsf{pad}_{i,2} \oplus \widehat{\mathsf{pad}}_{i,2}) = \widehat{\mathcal{H}}_i \oplus \mathcal{H}_i \tag{5.6}$$

$$\widehat{\mathsf{pad}}_{j,2} = \mathsf{pad}_{j,2} \oplus \Delta \tag{5.7}$$

We now argue that the output wire of $\widehat{g}_j$ is invalid. From an argument similar to case (1) of Lemma 5.2.3 (since the input wire keys to $\widehat{g}_j$ are valid), (5.7) imposes a constraint on the ciphertexts of $\widehat{g}_j$. Thus the probability that output wire key is valid is bounded by the probability of finding r_1 and r_2 such that $r_1 \oplus r_2$ is δ for some fixed δ and $r_1^{\ll} \oplus r_2^{\ll}$ collides with the output of the function H on a fixed value. By collision resistance of the function H, this happens only with probability $\leq 1/2^{\lambda}$.

By Lemma 5.2.3 and the union bound, we have that, $\Pr[\mathsf{De}(\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(x, \widehat{e})), d) \neq \perp] \leq |C|q^2/2^{\kappa}$, where $|C|$ is the number of gates in the circuit, and q is the number of queries to the function H that $\mathcal{A}$ is allowed to make.

Since the input x that lead to the above wire labels was arbitrary, we have that, given $\mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}), \mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$,

$$\forall x, \Pr[\mathsf{De}(\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(x, \widehat{e})), d) \neq \perp] = \mathsf{negl}(\kappa)$$

□

As calculated in the proof, the probability of hash collision is bounded by $|C|q^2/2^{\kappa}$. See Section 5.1 for discussion on parameter choices.

5.2.3 Hashing in half-gates garbling scheme

The current state of the art for garbled circuit construction is the half-gates scheme of Zahur et al. In the half-gates construction, two ciphertexts are used for each AND gate and the construction is compatible with the free-XOR technique [KS08a]. A half-gate is a garbled AND gate where one of the inputs to the gate is known in clear to one of the parties. Consider an AND gate $c = a \wedge b$. Now suppose the generator chooses a uniformly random bit r, and imagine we can have the evaluator learn the value of $r \oplus b$. We can write c as

$$c = a \wedge b = (a \wedge r) \oplus (a \wedge (r \oplus b))$$

[ZRE15] show how to garble the first AND gate with a generator-half-gate where the generator knows one of the values r, and the second AND gate with evaluator-half-gate since the evaluator know $r \oplus b$. The full AND gate is garbled by taking XOR of the two half-gates. Each garbled half-gate is one ciphertext, and with free-XOR, the full AND gate is two ciphertexts.

Let $\mathsf{GC}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Eval}')$ be the algorithms of the half-gate garbling procedure in [ZRE15]. The algorithms for encoding and evaluation in our scheme are the same; we only include the garbling and decoding algorithms, Gb and De. We assume that the half-gate garbling scheme outputs wire labels corresponding to both 0 and 1 on the output wires as the decoding information. Gb outputs a garbled circuit, the encoding and decoding information and the hash of the garbled circuit. De returns a decoded output or $\perp$ if the garbled output is invalid.

- $\mathsf{Gb}(1^\kappa, \mathcal{C})$: On input security parameter κ and a circuit $\mathcal{C}$, run the half-gate garbling algorithm $(e', d', \mathsf{GC}') \leftarrow \mathsf{Gb}'(1^\kappa, \mathcal{C})$, where $\mathsf{GC}' = \{\tau_{G_i}, \tau_{E_i}\}_{g_i \in G_{\mathsf{inter}}}$, and $d' = \{(d_i^0, d_i^1)\}_{W_i \in W_{\mathsf{out}}}$. Set the encoding information e, decoding information d, garbled circuit GC and hash $\mathcal{H}(\mathsf{GC})$ as

$$e = e', \quad d = d', \quad \mathsf{GC} = \mathsf{GC}', \quad \mathcal{H}(\mathsf{GC}) = \bigoplus_i (\tau_{G_i} \oplus \tau_{E_i})$$

- En is defined to be En' .
- Eval is defined to be Eval'.
- $\mathsf{De}(\mathbf{Y}, d)$: On input the decoding information d and the garbled output of the circuit $\mathbf{Y} = (Y_1, ..., Y_m)$, for each output wire i of the circuit $\mathcal{C}$, parse d as $d = \{d_i^0, d_i^1\}_{i \in [m]}$. Then, set $y_i = b$ if $Y_i = d_i^b$ and $y_i = \perp$ if $Y_i \notin \{d_i^0, d_i^1\}$. Output the result $\boldsymbol{y} = (y_1, ..., y_m)$ if $\forall i, y_i \neq \perp$. Else, output $\perp$.

Figure 5.2: The Half-Gate Free Hash garbling scheme $\mathsf{half}\mathcal{G}$

Note that in the construction of hashed garbling scheme for half-gates above, the hash is the XOR of all the ciphertexts. Unlike our construction for general garbled circuits (cf. Figure 5.1), we do not modify the wire keys. Since the garbled circuit is the same as the original half-gates construction, we retain the privacy and authenticity properties. To argue hash security, first observe that in the half-gates scheme both ciphertexts in a garbled gate (one per half-gate) are decrypted and used for output wire computation. Consider an attacker $\mathcal{A}$ which modifies a gate table and changes one entry to decrypt to a wrong label. Then there must be another modified entry to correct the hash, and *both* modified entries need to

decrypt correctly during evaluation to produce a valid label. Thus, in the half-gate garbling, the intuition for hash security is similar to that of our original 4-row construction. Namely, any modified gate will break the XOR hash. Further, any gate table that brings back the hash to the correct value will result in an invalid output wire label. We provide a proof sketch below.

Theorem 5.2.5. *The Half-Gate Free Hash garbling scheme* halfG *described in Figure 5.2 satisfies hash security as defined in Definition 5.2.1 assuming the collision-resistance of* H*.*

Proof Sketch. Given an adversary $\mathcal{A}$ who outputs $(\mathsf{GC}, e, \widehat{\mathsf{GC}}, \widehat{e}, d, h)$ such that $\mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}) = h$, $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$, we show that $\forall x, \Pr[\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(\widehat{e}, x)) \neq \perp] = \mathsf{negl}(\kappa)$. Since $\mathsf{GC} \neq \widehat{\mathsf{GC}}$, they must differ in at least one garbled gate, and let $g_i \neq \widehat{g}_i$ be the first gate in topological order that differs: $g_i = \{\tau_{G_i}, \tau_{E_i}\}$ and $\widehat{g}_i = \{\widehat{\tau}_{G_i}, \widehat{\tau}_{E_i}\}$. Let $\widehat{\mathcal{H}}_i$ be the running hash up until gate $\widehat{g}_i$ in $\widehat{\mathsf{GC}}$. We consider the following cases:

1. $\widehat{\mathcal{H}}_i = \mathcal{H}_i$ where $\mathcal{H}_i$ is the running hash until gate g_i in GC. Now $g_i \neq \widehat{g}_i$ and $\widehat{\mathcal{H}}_i = \mathcal{H}_i$ implies that both half-gates are modified since $\widehat{g}_i$ is the first gate that differs from GC. That is,

$$\tau_{G_i} \neq \widehat{\tau}_{G_i} \text{ and } \tau_{E_i} \neq \widehat{\tau}_{E_i}$$

   Let $(\widehat{K}_a, \widehat{K}_b)$ be the input wire keys of $\widehat{g}_i$. The output wire key of $\widehat{g}_i$ during Eval is given by

$$\widehat{K} = H(\widehat{K}_a) \oplus s_a \widehat{\tau}_{G_i} \oplus H(\widehat{K}_b) \oplus s_b(\widehat{\tau}_{E_i} \oplus \widehat{K}_a)$$

   where s_a and s_b are select bits. The probability that $\widehat{K}$ is valid is at most $1/2^\lambda$ by the collision resistance of function H. Now, by inductively using argument similar to cases (2) and (3) of Lemma 5.2.3, the wire keys of $\widehat{\mathsf{GC}}$ remain invalid.

2. $g_i \neq \widehat{g}_i$, $\widehat{\mathcal{H}}_i \neq \mathcal{H}_i$ and $\mathcal{H}(\mathsf{GC}) = \mathcal{H}(\widehat{\mathsf{GC}})$ implies there must be a gate $\widehat{g}_j \neq g_j$ such that

$$\widehat{\tau}_{G_j} \oplus \widehat{\tau}_{E_j} = \widehat{\mathcal{H}}_i \oplus \mathcal{H}_i \oplus (\tau_{G_j} \oplus \tau_{E_j}) \tag{5.8}$$

   We now argue that the output wire of $\widehat{g}_j$ is invalid: (5.8) imposes a constraint on the ciphertexts of $\widehat{g}_j$. Thus the probability that output wire key is valid is bounded by the probability of finding r_1 and r_2 such that $r_1 \oplus r_2$ is δ

for some fixed δ and r_1 and r_2 collide with the outputs of function H. By collision resistance of H, this happens with probability at most $1/2^\lambda$. Again, inductively all further wire keys of $\widehat{\mathsf{GC}}$ remain invalid.

By the union bound, we have that, $\Pr[\mathsf{De}(\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(\widehat{e}, x)), d) \neq \perp] \leq |C|q^2/2^\kappa$, where $|C|$ is the number of gates in the circuit, and q is the number of queries to the function H that $\mathcal{A}$ is allowed to make.

As calculated in the proof, the probability of hash collision is bounded by $|C|q^2/2^\kappa$. See Section 5.1 for discussion on parameter choices.

5.2.4 Performance and Impact

Cut-and-choose protocols using $\mathsf{h}\mathcal{G}$. As pointed out in [GMS08], an improvement in communication complexity can be achieved by taking the following approach. To compute a garbled circuit, the garbler P_1 generates a random PRG seed. Then the output of the pseudorandom generator is used as the random tape for the garbling algorithm. In C&C, P_1 sends to P_2 only a collision-resistant (CR) hash of each GC. In a later stage of the protocol, if a GC GC is chosen as a check circuit and needs to be opened, P_1 simply sends the seed corresponding to that circuit to P_2.

$\mathsf{h}\mathcal{G}$ hash can be used in C&C similarly to standard CR hash of GC. In [GMS08], P_1 commits via a collision resistant hash function to garbled circuits. These GCs can be either *good* or *cheating*. Importantly, due to the CR property of the hash, a malicious P_1 cannot change this designation at a later time. In using $\mathsf{h}\mathcal{G}$, P_1 has the same choice: he can compute $\mathsf{h}\mathcal{G}$ of either a good or a cheating GC. If he computed and sent the hash h of a good garbled circuit GC, then h cannot be claimed to match a cheating evaluation circuit $\widehat{\mathsf{GC}}$, even if the XOR hash $H(\mathsf{GC}) = H(\widehat{\mathsf{GC}})$. Indeed, w.h.p., evaluation of such a $\widehat{\mathsf{GC}}$ will fail and P_2 will abort, *independently* of P_2's input. Similarly, if P_1 computed and sent the hash of a cheating circuit $\widehat{\mathsf{GC}}$, it cannot be later opened as a good check circuit GC.

We stress that we must be careful when P_2 is allowed to abort, so as to not allow a selective failure attack. Specifically, a malicious P_1 could cause evaluation failure by sending an invalid label on a specific input wire/value pair or by generating a GC which produces an invalid label based on a value of an internal wire. Thus, while it is OK for P_2 to abort if it sees a GC which does not match the $\mathsf{h}\mathcal{G}$-hash, it should not (necessarily) abort simply based on seeing a decoding failure. Instead, this failure should be treated by the C&C procedure. We stress that it is protocol dependent, and protocol security should be evaluated. At the high level, our hashing guarantees that the garbler cannot open/equivocate an "honest" hashed circuit as a valid "malicious" circuit (or vice versa). However, he can open any (i.e. honest or malicious) hashed circuit as a "broken" one (i.e. one which will fail evaluation).

	Our $\mathsf{h}\mathcal{G}$ construction	Garble + SHA	justGarble
Standard Garbling	31.1	226.7	29
Half-gates	26.8	157.7	25.3

Table 5.1: Free Hash Implementation results

Covert C&C protocols [AL07, KM15], as well as C&C based on majority output, such as [LP11], can be made to work with $\mathsf{h}\mathcal{G}$. Indeed, exercising the extra power the adversary has (turning a good or bad evaluation circuit into a broken evaluation circuit) will simply cause covert evaluator to abort independently of its input. Similarly, in [LP11], the evaluation circuits which were made broken cannot be used to contribute to majority output. Using $\mathsf{h}\mathcal{G}$ with [KM15] requires a bit of care. [KM15] actually already explicitly support using [GMS08]. Using $\mathsf{h}\mathcal{G}$ differs from [GMS08] only in that a cheating P_1 can open an honest evaluation circuit as a broken one, resulting in an abort. However, the same effect could be achieved by P_1 sending an invalid signature on the garbled circuit.

We note that [Lin13] uses [LP11] as a basic step in cheating punishment and our $\mathsf{h}\mathcal{G}$ can be used within the [LP11] subprotocol of [Lin13]. However, it is not immediately clear $\mathsf{h}\mathcal{G}$ can be used elsewhere in [Lin13]. This is because the cheating punishment relies on evaluator having received a good evaluation circuit to recover the cheating garbler's input. However, in our case, malicious garbler can present a broken circuit, preventing input recovery.

Similarly, it is not immediately clear that the dual-execution C&C protocols of [HKE13, KMRR15] can take advantage of $\mathsf{h}\mathcal{G}$. Intuitively, this is because a malicious generator P_1 might produce a single cheating circuit, which is likely to be chosen for evaluation among a number of honest circuits. Then, P_1 will open all honest evaluation circuits as broken ones. Avoiding selective failure attack, P_2 will not abort, and the resulting output will depend on the output of the cheating circuit.

Implementation. We implemented our scheme using libgarble [Mal] for garbling and report on the performance below. In Table 5.1, we compare the cost of our GC hashing construction with garbling and then hashing the GC using SHA. We use the AES circuit to garble in the comparisons. The numbers in Table 5.1 are in cycles per gate. The configuration of the machine we used to run our implementation is: 2.3 GHz Core i5-2410M processor with 4 GB RAM. The processor has AES-NI integrated.

We believe that free hashing will simplify and speed up GC use particularly in larger systems using GC, such as the Blind Seer encrypted database [PKV+14,

FVK+15], where GC processing will be competing for the CPU resource with a number of other tasks.

SFE of private certified functions. We now consider the use case described in Section 1.3.2, where a Certificate Authority (CA) generates and certifies a number of GCs for use by the subscribers of the CA. In this case, clearly, CA is the bottleneck; Table 5.1 demonstrates over 6× performance improvement for the state-of-the-art half-gates GC, as compared with using standard hashing available with the OpenSSL library. Again, we stress that with half-gates hashing, simple XOR of all rows of all the gate tables provides a secure hash. This allows simple implementation in addition to the performance improvement.

Impact on Cut-and-choose. We discuss the SFE performance improvement brought by our work on the example of the state-of-the-art approach of [LP11] and [KM15]. (Subsequent improvements to [LP11], as well as C&C, covert and other GC protocols will benefit from free GC hashing correspondingly). We review the C&C choices and parameters of [LP11, AO12, KM15] in light of [GMS08] and free hashing allowed by our work. We will show that:

1. Computing and sending additional GC hashes does not increase communication cost (computation cost is minimal due to our work), but significantly reduces cheating probability (see Table 5.2).

2. Keeping the cheating probability constant, we improve total C&C time by 43−64% by sending circuit hashes instead of circuits as suggested by [GMS08] (See Table 5.3).

For concreteness, to achieve a cheating probability of, say, 2^{-40}, the number of garbled circuits that need to be sent is n. This incurs a communication cost, in bits, of k, where $k = n\mathsf{C}$, and C is the cost of a garbled circuit.

Sending only the hashes of the garbled circuits in the beginning of the cut-and-choose, let the total number of garbled circuits be $\tilde{n}$. Let h be the size of the hash of a GC, which is the communication cost of a check circuit. Now, we have that the communication bits incurred,

$$\tilde{k} = \tilde{n}\mathsf{h} + \frac{1}{2}\tilde{n}\mathsf{C}$$

Setting the communication complexity to be the same, $\tilde{k} = k = n\mathsf{C}$, we have,

$$n = \tilde{n}\mathsf{q} + \frac{\tilde{n}}{2}$$

where $\mathsf{q} = \frac{\mathsf{h}}{\mathsf{C}}$ is the ratio of the cost of a check circuit and the cost of a garbled circuit. For $q < 1/2$, we have $\tilde{n} = \frac{n}{q+\frac{1}{2}} > n$, thus giving a cheating probability

$2^{-\tilde{n}} < 2^{-n}$ for the same communication complexity. For large circuits, we expect $\mathsf{C} \gg \mathsf{h}$, giving concrete improvements in the security at no additional communication cost.

	Communication k	Number of circuits	Cheating probability / Deterrence
[LP11]	$k = 125\|GC\|$	$n = 125$	2^{-40}
[LP11] with $\mathsf{h}\mathcal{G}$, $q = 1/4$	$k = 125\|GC\|$	$\tilde{n} = 166$	2^{-51}
[LP11] with $\mathsf{h}\mathcal{G}$, $q = 1/8$	$k = 125\|GC\|$	$\tilde{n} = 200$	2^{-62}
[KM15] without [GMS08]	$k = 10\|GC\|$	$n = 10$	0.9
[KM15] with $\mathsf{h}\mathcal{G}$, $q = 1/4$	$k = 10\|GC\|$	$\tilde{n} = 36$	0.972
[KM15] with $\mathsf{h}\mathcal{G}$, $q = 1/8$	$k = 10\|GC\|$	$\tilde{n} = 72$	0.986

Table 5.2: SFE Performance improvement using Free Hash

We note that [KM15] incorporates the [GMS08] hashing in the protocol. As we discussed, sending circuits over a fast channel may only be about 3× slower than hardware-assisted garbling, while computing SHA1 may be up to 6× slower than such garbling. Hence, sending circuits over a fast channel may actually be faster than generating SHA1 hash. Therefore, in our calculations for the fast channel setting as above, we consider [KM15] without [GMS08] hash. For comparison, we note that the protocol of [Lin13] achieves covert security with deterrent $\epsilon = 0.999$ using 11 circuits, but does not achieve public verifiability.

Performance improvement for constant cheating probability. Consider the task of evaluating a billion-gate circuit (cf. [KSS12]). We show estimated improvement due to our technique as applied to [LP11] and [KM15]. We do this in terms of expended time by unifying the computation and communication costs of generating and sending garbled circuits. These calculations are not based on specific implementations or protocol definitions. Instead they are based on simple estimates of time needed to generate, hash and send GCs, and adding them together.

We first calculate and explain the computation and communication costs in seconds of our basic tasks.

According to [BHKR13], using JustGarble to garble the AES circuit (6660 non-XOR gates) takes 637 microseconds. Adjusting for size, we calculate that the time taken for GC generation for a circuit with 1 billion gates to be 95 seconds. For communication, assuming ideal scenario in 1Gbps channel, assume we can send 1 billion bits/sec. Thus the time to send a circuit of 1 billion gates is 256 seconds at (assuming half gates and 2×128 bits per gate).

The total number of seconds needed in the cut-and-choose phase to maliciously evaluate a 1 billion-gate circuit with 2^{-40} cheating probability using previous technique and our construction using the optimal parameters. In our calculation we include the costs of generating, hashing (in our scheme) and sending the GCs. We do not include the cost of *regenerating the check circuits* at the evaluator's end that

	Total number of circuits	Number of check circuits	Circuits sent	Time (in secs)
[LP11]	125	75	125	43875
[LP11] $+\mathsf{h}\mathcal{G}$	125	75	50	24675

Table 5.3: A billion-gate circuit. Execution time estimates of cut-and-choose with our improvements to achieve cheating probability of 2^{-40}

	Total number of circuits	Number of check circuits	Circuits sent	Time (in secs)
[AL07]	10	9	10	3510
[AL07]$+\mathsf{h}\mathcal{G}$	10	9	1	1260
[KM15] without [GMS08]	10	9	10	3510
[KM15]$+\mathsf{h}\mathcal{G}$	10	9	1	1260

Table 5.4: A billion-gate circuit. Execution time estimates of cut-and-choose with our improvements to achieve deterrence of $\epsilon = 0.9$.

is incurred by our technique. This is because this cost is also incurred by other techniques. Indeed, checking correctness of a circuit that the evaluator already has (directly, or when using [GMS08] hash) is simplest and fastest by receiving its generating seed, reconstructing and comparing. We are concerned only with the cut-and-choose phase, and ignore the time taken for OT and GC evaluation in the protocol and show how our construction allow for reduced execution time in the cut-and-choose phase.

The cost in seconds calculated in Table 5.3 is obtained by adding the time to generate, hash (if needed) and send all the required garbled circuits. As explained above, we assume that it takes 95 seconds to generate a 1-Billion gate GC, and 256 seconds to send it.

5.3 Application to Zero-Knowledge

Ishai et al. [IKOS07, IKOS09] introduced the "MPC-in-the-head" technique that allows a generic transformation of an MPC protocol into a zero-knowledge proof. This provides a powerful tool to obtain black-box constructions for generic statements without relying on expensive Karp reductions. Recently, this technique was further studied in [GMO16] resulting in efficient ZK arguments tailored

for Boolean circuits. They study variants of the "MPC-in-the-head" framework, plug in different MPC protocols, and provide concrete estimates of soundness. In [HV16], the authors extend this idea, into "2PC-in-the-head" and give a generic transformation from a secure two-party computation protocol to a ZK proof.

We present a protocol that is essentially a special case of general cut-and-choose. Since the verifier has no input, we do not have to handle selective failure where the evaluator's abort could leak a bit of his input, or ensure input consistency of the garbler, since the circuit is evaluated on an input entirely known to the garbler. While in [HV16], a similar approach is seen as "2PC-in-the-head", we cast our protocol as a cut-and-choose protocol. Standard transformations based on Oblivious Transfer and Fiat-Shamir may then be applied to obtain a 2-round ZK and a NIZK respectively. Loosely speaking, choosing to reveal P_1's view in "2PC-in-the-head" in [HV16] is equivalent to choosing a circuit to be a check circuit in our protocol; and choosing to reveal P_2's view corresponds to a circuit being an evaluation circuit. We make some observations about the protocol that could lead to improved efficiency.

- We do not need the gadgets for input consistency checks of the prover, and input recovery mechanisms in case of inconsistent outputs as used in several works [Lin13, LR14, RR16, MR17]. We note that we do not need to enforce output recovery when two evaluated circuits result in different outputs. The output recovery mechanism that is used in general 2PC protocols [Lin13, LR14, LR15, AMPR14, MR17] relies on authenticity property of the underlying garbling scheme. This means that taking the [GMS08] approach of hashing GCs can benefit the sigma protocol to achieve an improvement in communication complexity. The weaker hash definition introduced in this chapter will suffice for our protocol and Free Hash allows this improvement at no additional computational overhead.

- We do not rely on the authenticity property of the underlying garbling scheme. It remains open to study authenticity-free garbled circuits and the possibility of taking advantage of this, in a vein similar to privacy-free garbled circuits [FNO15] and their application to zero-knowledge.

Sigma protocol from Garbled Circuits. Following the standard cut-and-choose paradigm, the prover constructs many independent garbled circuits of the circuit implementing C. The verifier generates a uniformly random string as the challenge. The challenge bits determine whether a garbled circuit is to be opened and checked, or is to be evaluated. If all the check circuits are valid and if all evaluated circuits output 1, the verifier accepts. This is sufficient for our application since there is no need to address selective failure issues that arise in general

computation. This is because the verifier has no private input, and hence there is nothing to leak when he aborts.

We denote the garbled input corresponding to the jth wire in the ith circuit by X_{ij}^b, for $b \in \{0,1\}$. Our Σ-protocol is given in Figure 5.3.

We now prove that the protocol in Figure 5.3 is a Σ-protocol by proving that it is honest-verifier zero knowledge and 2-special sound. The 2-special soundness implies a soundness error of $2^{-\mu}$ where μ is the length of the challenge, which is the number of garbled circuits. It is easy to see that the protocol is 3-move and complete.

Theorem 5.3.1. *Protocol in Figure 5.3 is a Σ-protocol for the relation R_f with 2-special soundness, assuming $\mathcal{G}$ is a* correct, private, verifiable *garbling scheme.*

Proof.

1. Special honest verifier zero-knowledge: The simulator Sim on input x and r does the following. For every bit r_i such that $r_i = 0$, Sim constructs the corresponding garbled circuits GC_i honestly for the function C. For every bit r_i such that $r_i = 1$, Sim uses the garbled circuit simulator that exists for a private garbling scheme, to construct a fake garbled circuit that always outputs 1. Indistinguishability of the simulator's output from the real transcript follows from the privacy of the garbling scheme.

2. Special soundness: Consider two accepting transcripts $\tau_1 = (a, r^1, e^1), \tau_2 = (a, r^2, e^2)$. Now, $r^1 \neq r^2$ means there must exist some index i where the two strings differ, that is, $r_i^1 \neq r_i^2$. w.l.o.g, this means that, the ith garbled circuit in a is a check circuit in τ_1 and an evaluation circuit in τ_2. Since both transcripts are accepting, GC_i is a valid garbled circuit that was opened and checked in τ_1 and GC_i evaluated to 1 in τ_2. We can now extract the input bits by mapping the input wire keys of the garbled circuit GC_i in τ_2 with the opened garbled circuit GC_i in τ_1. By correctness and verifiability of the garbling scheme used, the extracted bits x_i are such that $R(z, x) = 1$.

□

Figure 5.3: The GC based Σ-protocol

Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Eval}, \mathsf{Ve})$ be a verifiable garbling scheme. Let C be the circuit implementing the relation $R(x, w)$. Both parties have as input the security parameter κ, the statistical security parameter μ and instance x, and the prover additionally has as input a witness w such that $R(x, w) = 1$.

- **Commit:**

  1. The prover P constructs μ independent garbled circuits $\mathsf{GC}_1, \cdots, \mathsf{GC}_\mu$ for C.
  $$(\mathsf{GC}, \{X_j^0, X_j^1\}_{j\in[n]}, \{Z^0, Z^1\}) \leftarrow \mathsf{Gb}(1^\kappa, C)$$
  2. The prover sends the first message to the verifier.
  $$a = \{(\mathsf{GC}_1, \{Z_1^0, Z_1^1\}), \ldots, (\mathsf{GC}_\mu, \{Z_\mu^0, Z_\mu^1\})\}$$

- **Prove:**

  1. The verifier chooses a uniformly random string r of length μ.
  $$r \leftarrow \{0,1\}^\mu$$
  The verifier sends the challenge r to the prover.
  2. For the challenge string r, if bit $r_i = 0$, the garbled circuit GC_i is a check circuit and if $r_i = 1$, GC_i is an evaluation circuit. The prover reveals the randomness used to construct GC_i if $r_i = 0$, and reveals the input wire keys corresponding to input x if $r_i = 1$.
  $$e_i = \begin{cases} \{X_{ij}^{w_j}\}_{j=1}^n & \text{if } r_i = 1 \\ \{X_{ij}^0, X_{ij}^1\}_{j=1}^n & \text{otherwise} \end{cases}$$
  Send the response $e = \{e_i\}_{i=1}^\mu$ to the verifier.

- **Verify:**

  1. If $\mathsf{Ve}(C, \mathsf{GC}_i, \{X_{ij}^0, X_{ij}^1\}_{j=1}^n, \{Z_i^0, Z_i^1\}) = 0$ for some i with $r_i = 0$, the verifier outputs reject and aborts.
  2. If $\mathsf{De}(\mathsf{Eval}(\mathsf{GC}_i, \{X_{ij}^{x_j}\}_{j=1}^n), \{Z_i^0, Z_i^1\}) \neq 1$ for some i with $r_i = 1$, the verifier outputs reject and aborts.
  3. If the verifier did not abort, she outputs accept.

Chapter 6

Conclusion

In this dissertation, we constructed *efficient* zero-knowledge proofs for combination statements that have both algebraic and non-algebraic components. We showed how to combine state-of-the-art approaches of sigma protocols, garbled circuits and SNARKs while retaining the efficiency of each of them. We also showed applications of zero-knowledge for combinations statements in anonymous credentials and privacy-preserving proof of solvency for Bitcoin. Further, we proposed a definition for hash security of garbled circuits, and gave constructions.

Several interesting questions remain open. We summarize some of them below.

- It is a natural direction to explore how the techniques we introduced to combine GC and sigma protocols might work with the "MPC-in-the-head" technique. Recent work of ZKBoo [GMO16] that uses "MPC-in-the-head" gives efficient sigma protocols to prove non-algebraic statements. Exploring ZKBoo together with algebraic sigma protocols might lead to non-interactive proofs for combination statements with different trade-offs.

- It remains to study authenticity-free garbled circuits. It is known that giving up on privacy of garbled circuits leads to more efficient constructions and they suffice for application in certain zero-knowledge protocols. Besides being interesting to separate the different garbled circuit properties, the feasibility of efficient authenticity-free garbled circuits could lead to similar application to GC-based zero-knowledge protocols.

- Despite many advances in the efficiency of Garbled Circuit constructions, the techniques mostly apply to boolean circuits. In our first construction combining garbled circuits and sigma protocols for proof of a combination of algebraic and non-algebraic statement, the bottleneck in terms of efficiency is the garbling of a multiplication circuit. This can be greatly improved if we could garble an arithmetic circuit directly. In a recent work, [BMR16] present simple new constructions that apply to arithmetic circuits. It is interesting to

try to use these gadgets for arithmetic garbling with our technique for zero-knowledge proofs. It also remains to study if we can get better privacy-free garbled circuits in the arithmetic world.

Bibliography

[AGM17] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for compound statements. Preprint, 2017.

[Aho87] Alfred Aho, editor. *19th ACM STOC*. ACM Press, May 1987.

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, Heidelberg, February 2007.

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014.

[AO12] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 681–698. Springer, Heidelberg, December 2012.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In IEEE S&P 2014 [IEE14], pages 459–474.

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374. Springer, Heidelberg, March 2008.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In IEEE S&P 2013 [IEE13], pages 478–492.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, October 2012.

[bit] Technical background of version 1 bitcoin addresses. https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses.

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Sadeghi et al. [SGY13], pages 1087–1098.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM, 1990.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In Weippl et al. [WKK+16], pages 565–577.

[BOGG+90] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 37–56. Springer, Heidelberg, August 1990.

[BÖS11] Joppe W. Bos, Onur Özen, and Martijn Stam. Efficient hashing using the AES instruction set. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 507–522. Springer, Heidelberg, September / October 2011.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 431–444. Springer, Heidelberg, May 2000.

[BR93a] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[BR93b] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *Advances in Cryptology-Eurocrypt'96*, pages 399–416. Springer, 1996.

[Bra99] Stefan Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy*. PhD thesis, Eindhoven Institute of Technology, Eindhoven, The Netherlands, 1999.

[Bra13] Luís T. A. N. Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique - (extended abstract). In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 441–463. Springer, Heidelberg, December 2013.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335. Springer, Heidelberg, August 2002.

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 234–252. Springer, Heidelberg, December 2008.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols.

In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *26th FOCS*, pages 372–382. IEEE Computer Society Press, October 1985.

[CG13] Ran Canetti and Juan A. Garay, editors. *CRYPTO 2013, Part II*, volume 8043 of *LNCS*. Springer, Heidelberg, August 2013.

[CGM16] Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 499–530. Springer, Heidelberg, August 2016.

[Cha86] David Chaum. Showing credentials without identification: Signatures transferred between unconditionally unlinkable pseudonyms. In Franz Pichler, editor, *EUROCRYPT'85*, volume 219 of *LNCS*, pages 241–244. Springer, Heidelberg, April 1986.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 1205–1216. ACM Press, November 2014.

[CPS08] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In Wagner [Wag08], pages 1–20.

[Cré90] Claude Crépeau. Verifiable disclosure of secrets and applications (abstract). In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 150–154. Springer, Heidelberg, April 1990.

[CS97a] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology-CRYPTO'97*, pages 410–424. Springer, 1997.

[CS97b] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Kaliski Jr. [Kal97], pages 410–424.

[CvT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 110–123. Springer, Heidelberg, August 1995.

[CZ09] Jan Camenisch and Gregory M Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security*, pages 108–127. Springer, 2009.

[Dam] Ivan Damgård. On Σ-protocols. http://www.cs.au.dk/~ivan/Sigma.pdf.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *Advances in Cryptology-EUROCRYPT 2000*, pages 418–430. Springer, 2000.

[DBB+15] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In Ray et al. [RLK15], pages 720–731.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology-ASIACRYPT 2002*, pages 125–142. Springer, 2002.

[DLFKP16] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *2016 IEEE Symposium on Security and Privacy*, pages 235–254. IEEE Computer Society Press, May 2016.

[FFS87] Uriel Feige, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In Aho [Aho87], pages 210–217.

[FGK17] Xiong Fan, Chaya Ganesh, and Vladimir Kolesnikov. Hashing garbled circuits for free. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications*

of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III, pages 456–485, 2017.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Oswald and Fischlin [OF15], pages 191–219.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Kaliski Jr. [Kal97], pages 16–30.

[FOC86] *27th FOCS*. IEEE Computer Society Press, October 1986.

[For87] Lance Fortnow. The complexity of perfect zero-knowledge (extended abstract). In Aho [Aho87], pages 204–209.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Odlyzko [Odl87], pages 186–194.

[FVK+15] Ben A. Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M. Bellovin. Malicious-client security in blind seer: A scalable private DBMS. In *2015 IEEE Symposium on Security and Privacy*, pages 395–410. IEEE Computer Society Press, May 2015.

[GG14] Juan A. Garay and Rosario Gennaro, editors. *CRYPTO 2014, Part II*, volume 8617 of *LNCS*. Springer, Heidelberg, August 2014.

[GGO+] Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Sean Gulley, and Wajdi Feghali. Improving OpenSSL performance. https://software.intel.com/sites/default/files/open-ssl-performance-paper.pdf. Retrieved Feb 3, 2017.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 1069–1083, 2016.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

[GMS08] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In Smart [Sma08], pages 289–306.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In FOCS 1986 [FOC86], pages 174–187.

[GMW87a] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Aho [Aho87], pages 218–229.

[GMW87b] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Odlyzko [Odl87], pages 171–185.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 123–128. Springer, Heidelberg, May 1988.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Smart [Sma08], pages 415–432.

[HKE13] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In Canetti and Garay [CG13], pages 18–35.

[HKK+14] Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. In Garay and Gennaro [GG14], pages 458–475.

[HKT11] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 89–98. ACM Press, June 2011.

[HV16] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. On the power of secure two-party computation. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 397–429. Springer, Heidelberg, August 2016.

[ide10] Specification of the identity mixer cryptographic library (revised version 2.3.0). Technical Report RZ 3730, IBM Research, April 2010.

[IEE13] *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013.

[IEE14] *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Sadeghi et al. [SGY13], pages 955–966.

[JS07] Stanisław Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology-EUROCRYPT 2007*, pages 97–114. Springer, 2007.

[Kal97] Burton S. Kaliski Jr., editor. *CRYPTO'97*, volume 1294 of *LNCS*. Springer, Heidelberg, August 1997.

[KKL+16] Vladimir Kolesnikov, Hugo Krawczyk, Yehuda Lindell, Alex J. Malozemoff, and Tal Rabin. Attribute-based key exchange with general policies. In Weippl et al. [WKK+16], pages 1451–1463.

[KKW16] W. Sean Kennedy, Vladimir Kolesnikov, and Gordon Wilfong. Overlaying circuit clauses for secure computation. Cryptology ePrint Archive, Report 2016/685, 2016. http://eprint.iacr.org/2016/685.

[KM15] Vladimir Kolesnikov and Alex J. Malozemoff. Public verifiability in the covert model (almost) for free. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 210–235. Springer, Heidelberg, November / December 2015.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Garay and Gennaro [GG14], pages 440–457.

[KMRR15] Vladimir Kolesnikov, Payman Mohassel, Ben Riva, and Mike Rosulek. Richer efficiency/security trade-offs in 2PC. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 229–259. Springer, Heidelberg, March 2015.

[Knu69] Donald E Knuth. The art of computer programming. vol. 2: Seminumerical algorithms. addisonwesley. *Reading, MA*, pages 229–279, 1969.

[Kol05] Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 136–155. Springer, Heidelberg, December 2005.

[KS06] Mehmet Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of yaos garbled circuit construction. In *27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

[KS08a] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

[KS08b] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming*, pages 486–498. Springer, 2008.

[KS08c] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In

Gene Tsudik, editor, *FC 2008*, volume 5143 of *LNCS*, pages 83–97. Springer, Heidelberg, January 2008.

[KS16] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 699–728. Springer, Heidelberg, May 2016.

[KSS09] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security*, pages 1–20. Springer, 2009.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 14–14, Berkeley, CA, USA, 2012. USENIX Association.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Canetti and Garay [CG13], pages 1–17.

[Lin15] Yehuda Lindell. An efficient transform from sigma protocols to nizk with a crs and non-programmable random oracle. In *Theory of Cryptography*, pages 93–109. Springer, 2015.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

[LMS16] Helger Lipmaa, Payman Mohassel, and Saeed Sadeghian. Valiant's universal circuit: Improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017, 2016. http://eprint.iacr.org/2016/017.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, Heidelberg, March 2011.

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Garay and Gennaro [GG14], pages 476–494.

[LR15] Yehuda Lindell and Ben Riva. Blazing fast 2PC in the offline/online setting with security for malicious adversaries. In Ray et al. [RLK15], pages 579–590.

[Mal] Alex J. Malozemoff. libgarble: garbling library based on justgarble. https://github.com/amaloz/libgarble.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[MR13] Payman Mohassel and Ben Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In Canetti and Garay [CG13], pages 36–53.

[MR17] Payman Mohassel and Mike Rosulek. Non-interactive secure 2pc in the offline/online and batch settings. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, pages 425–455, 2017.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, pages 275–292, 2005.

[NMT] Shen Noether, Adam Mackenzie, and Monero Core Team. Ring confidential transactions. https://lab.getmonero.org/pubs/MRL-0005.pdf.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.

[Odl87] Andrew M. Odlyzko, editor. *CRYPTO'86*, volume 263 of *LNCS*. Springer, Heidelberg, August 1987.

[OF15] Elisabeth Oswald and Marc Fischlin, editors. *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*. Springer, Heidelberg, April 2015.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in CryptologyCRYPTO91*, pages 129–140. Springer, 1991.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In IEEE S&P 2013 [IEE13], pages 238–252.

[PKV+14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. Blind seer: A scalable private DBMS. In IEEE S&P 2014 [IEE14], pages 359–374.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009.

[PZ13] C. Paquin and G. Zaverucha. U-prove cryptographic specification v1.1 (revision 2). Available online: www.microsoft.com/uprove, 2013.

[Rab77] Michael O. Rabin. Digitalized signatures. Foundations of secure computation. In *Richard AD et al. (eds): Papers presented at a 3 day workshop held at Georgia Institute of Technology, Atlanta*, pages 155–166. Academic, New York, 1977.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. http://eprint.iacr.org/2005/187.

[RLK15] Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors. *ACM CCS 15*. ACM Press, October 2015.

[RR16] Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 297–314, 2016.

[RS08a] Phillip Rogaway and John P. Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In Wagner [Wag08], pages 433–450.

[RS08b] Phillip Rogaway and John P. Steinberger. Security/efficiency tradeoffs for permutation-based hashing. In Smart [Sma08], pages 220–236.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[sec] Secp256k1. https://en.bitcoin.it/wiki/Secp256k1.

[SGY13] Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors. *ACM CCS 13*. ACM Press, November 2013.

[Sma08] Nigel P. Smart, editor. *EUROCRYPT 2008*, volume 4965 of *LNCS*. Springer, Heidelberg, April 2008.

[sS11] abhi shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 386–405. Springer, Heidelberg, May 2011.

[Sta96] Markus Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology-EUROCRYPT96*, pages 190–199. Springer, 1996.

[Vad99] Salil Pravin Vadhan. *A study of statistical zero-knowledge proofs*. PhD thesis, Massachusetts Institute of Technology, 1999.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In *STOC*, pages 196–203, New York, NY, USA, 1976. ACM Press.

[Wag08] David Wagner, editor. *CRYPTO 2008*, volume 5157 of *LNCS*. Springer, Heidelberg, August 2008.

[Wil] Zooko Wilcox. Proving bitcoin reserves. https://iwilcox.me.uk/2014/proving-bitcoin-reserves.

[Win84] Robert S Winternitz. A secure one-way hash function built from des. In *Security and Privacy, 1984 IEEE Symposium on*, pages 88–88. IEEE, 1984.

[WKK+16] Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors. *ACM CCS 16*. ACM Press, October 2016.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In FOCS 1986 [FOC86], pages 162–167.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Oswald and Fischlin [OF15], pages 220–250.