FORMALISING CRYPTOGRAPHY USING CRYPTHOL

DAVID THOMAS BUTLER

Doctor of Philosophy
School of Informatics
University of Edinburgh
2020

ABSTRACT

Security proofs are now a cornerstone of modern cryptography. Provable security has greatly increased the level of rigour of the security statements, however proofs of these statements often present informal or incomplete arguments. In fact, many proofs are still considered to be unverifiable due to their complexity and length. Formal methods offers one way to establish far higher levels of rigour and confidence in proofs and tools have been developed to formally reason about cryptography and obtain machine-checked proof of security statements.

In this thesis we use the CryptHOL framework, embedded inside Isabelle/HOL, to reason about cryptography. First we consider two fundamental cryptographic primitives: Σ-protocols and Commitment Schemes. Σ-protocols allow a Prover to convince a Verifier that they know a value x without revealing anything beyond that the fact they know x. Commitment Schemes allow a Committer to commit to a chosen value while keeping it hidden, and be able to reveal the value at a later time. We first formalise abstract definitions for both primitives and then prove multiple case studies and general constructions secure. A highlight of this part of the work is our general proof of the construction of commitment schemes from Σ-protocols. This result means that within our framework for every Σ-protocol proven secure we obtain, for free, a new commitment scheme that is secure also. We also consider compound Σ-protocols that allow for the proof of AND and OR statements. As a result of our formalisation effort here we are able to highlight which of the different definitions of Σ-protocols from the literature is the correct one; in particular we show that the most widely used definition of Σ-protocols is not sufficient for the OR construction. To show our frameworks are usable we also formalise numerous other case studies of Σ-protocols and commitment schemes, namely: the Σ-protocols by Schnorr, Chaum-Pedersen, and Okamoto; and the commitment schemes by Rivest and Pedersen.

Second, we consider Multi-Party Computation (MPC). MPC allows for multiple distrusting parties to jointly compute functions over their inputs while keeping their inputs private. We formalise frameworks to abstractly reason about two party security in both the semi-honest and malicious adversary models and then instantiate them for numerous case studies and examples. A particularly important two party MPC protocol is *Oblivious Transfer* (OT) which, in its simplest form, allows the Receiver to choose one of two messages from the other party, the Sender; the Receiver learns nothing of the other message held by the sender and the Sender does not learn which message the Receiver chose. Due to OTs fundamental importance we choose to focus much of our formalisation here, a highlight of this section of our work is our general proof of security of a *1-out-of-2 OT* (OT^1_2) protocol in the semi-honest model that relies on Extended Trapdoor Permutations (ETPs). We formalise the construction assuming only that an ETP exists meaning any instantiations for known ETPs only require one to prove that it is in fact an ETP — the security results on the protocol come for free. We demonstrate this by showing how the RSA collection of functions meets the definition of an ETP, and thus show how the security results are obtained easily from the general proof. We also provide proofs of security for the Naor Pinkas (OT^1_2) protocol in the semi-honest model as well as a proof that shows security for the two party GMW protocol — a protocol that allows for the secure computation of any boolean circuit. The malicious model is more complex as the adversary can behave arbitrarily. In this setting we again consider an OT^1_2 protocol and prove it secure with respect to our abstract definitions.

PUBLICATIONS

All of this thesis has been published in the papers listed below. While the papers below contain authors that are not David Butler he is the main, and first author, on all the publications associated with this thesis.

Peer Reviewed Conferences and Journals:

- David Butler, David Aspinall, and Adrià Gascón. 2017. How to Simulate It in Isabelle: Towards Formal Proof for Secure Multi-Party Computation. In ITP (Lecture Notes in Computer Science), Vol. 10499. Springer, 114–130.
- David Butler, David Aspinall, and Adrià Gascón. 2019. On the Formalisation of Σ-Protocols and Commitment Schemes. In POST (Lecture Notes in Computer Science), Vol. 11426. Springer, 175–196.
- David Butler, David Aspinall and Adrià Gascón. Formalising Oblivious Transfer in the Semi-Honest and Malicious Model in CryptHOL. In: CPP. ACM, 2020, pp. 229–243.

Under review in the Journal of Automated Reasoning:

- David Butler, Andreas Lochbihler, David Aspinall and Adrià Gascón, Formalising Σ-Protocols and Commitment Schemes using CryptHOL, Cryptology ePrint Archive, Report 2019/1185, 2019.

Archive of Formal Proof:

- David Butler and Andreas Lochbihler, Sigma Protocols and Commitment Schemes, Archive of Formal Proof, 2019.
- David Butler and David Aspinall, Multi-Party Computation, Archive of Formal Proof, 2019.

ACKNOWLEDGMENTS

My PhD has been a wonderful journey and I am sad it has ended. Along the way, I have had lots of much-needed support, guidance and friendship.

Firstly I would like to thank my supervisors, David Aspinall and Adrià Gascón, for supporting me throughout my PhD and giving me the freedom to work on the problems I found most interesting. Moreover you have given me the chance to meet and work with many other fantastic people. Thank you.

Much of the work I have done would not have been possible without Andreas Lochbihler. Andreas, I have very much enjoyed working together, I have learnt so much from our interactions and I am indebted to you for the time you have spent on our collaboration. My only regret is we did not work together sooner!

I would also like to thank Markulf Kohlweiss, with whom I have had many discussions fleshing out the low level details of "simple" paper proofs. Markulf, your understanding of how to "decode" a cryptographic proof majorly helped me on my way!

I am also incredibly grateful to Ewen Denney for giving me the chance to work with the Intelligent Systems Division at NASA Ames Research Center in the summer of 2018. And to Iain Whiteside for all that I have learned while we worked together there, and since.

To my friends at the Turing, often we have had little academically in common but your friendship and advice has in its own way contributed to this thesis greatly: Thibaut, FX, Alex, Daniel and more, thank you.

Outside of my academic work, I would like to thank those who have made the last years full of entertainment and competition on the Fives court: in particular Sam, Theo, Matt, you are all legends.

Lastly, I would like to thank my family. Whilst my research is most likely still a mystery to them they have always been there to listen to my worries and provide encouragement.

CONTENTS

DECLARATION

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

February 2020

David Thomas Butler

1 PRELIMINARIES

In this chapter we first motivate this thesis by providing evidence for the need to formalise cryptography. We then informally introduce the two areas whose intersection this thesis lies in, namely: cryptography (in particular provable security) and theorem proving.

1.1 MOTIVATION FOR THE THESIS

Since the emergence of provable security (we discuss what we mean by this in Section 1.2), security proofs have become a cornerstone of modern cryptography. Provable security allows for *reduction based* security arguments and has greatly increased the level of rigour of the security statements compared with the oftentimes informal and intuitive arguments given previously. The authority of such proofs, however, has been questioned [46] partially due to the proofs often presenting informal or incomplete arguments. In fact, many proofs are still considered to be unverifiable [42], or as Bellare and Rogaway remarked [11]:

> In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor.

A classic and well cited example of provable security failing the cryptographic community is the OAEP (Optimal Asymmetric Encryption) scheme that was introduced by Bellare and Rogaway in 1994 along with a reductionist security proof [10]. As a result of this work the scheme was included in the SET (Secure Electronic Transaction) electronic payment standard of MasterCard and Visa. However in 2000 Shoup found a flaw in the proof of security [70] that he proved could not be fixed.

Proofs of security are often long, complex and technical. This, combined with the often insufficient review process (the community tends to publish in conferences rather than journals), can often lead to problems. Moreover, due to the nature of the work, these problems oftentimes have dire consequences.

Many solutions have been proposed, attempting to alleviate provable security of its problems. For example structuring proofs and security properties into *games* or *simulations* have had some success in "taming the complexity" [71] of proofs. Such structures include game-based proofs [11, 71] where the adversary is challenged by a benign entity to output some knowledge it can only gain by somewhat breaking the desired security property, Canetti's universal composibility (UC) framework [26] whereby a protocol is secure if it can be simulated in an ideal environment and Maurer's constructive cryptography [55] a similar paradigm for proving security that allows for abstract theorems to be proven at an abstract level and inherited by lower level instantiations if the corresponding postulated axioms are met.

The approaches listed above provide a more formal footing for provable security. However Halevli's call for increased rigour [42] highlighted that machine checked proofs would be desirable to cryptographers to further reduce the problems associated with provable security. Formal, machine checked proof would expel any intuitive or informal arguments as well as reduce mistakes. The work contained in this thesis attempts exactly this, to provide fully machine checked proofs for public key cryptography.

We believe machine checked proof goes a long way in increasing the confidence of security proofs. There have been successes already; CryptoVerif [12], CertiCrypt [5], EasyCrypt [6], FCF [64] and CryptHOL [8, 50] are all tools that have been used to formalise various cryptographic constructions; from concrete constructions such as ElGamal encryption or Searchable Symetric Encryption (SSE) to instances of general frameworks such as UC and Constructive Cryptography. In this thesis we add to the literature of formal proofs by considering the fast growing area of MPC and two important low level primitives Σ-protocols and Commitment Schemes.

1.2 PROVABLE SECURITY, MPC, Σ-PROTOCOLS AND COMMITMENT SCHEMES

Like many areas of research the idea of provable security has evolved over time and there is not one obvious original text. Here we give a flavour of what is meant by this notion in the context of our work, and then introduce the areas of cryptography we concern ourselves with.

Provable security refers to defining properties pertaining to the security of cryptographic constructions and then providing proof that such constructions meet the definitions. More formally, an *adversary model* is constructed whereby the adversary is given some powers, for example, we might assume the adversary has access to certain parts of a system or can read or query certain encryptions. We also assume the adversary is feasible i.e. it runs in polynomial time. Using the adversary model one can define the desired security requirements. These usually take the form of requiring the adversary cannot do something, for example, guess which message was encrypted.

At the heart of provable security is the *reduction* argument — often used to prove a security property is satisfied. Informally a reduction based proof is constructed as follows: if an adversary can break the security property then, we show we can feasibly construct a new adversary that can break a *known hard problem*. Thus we have shown that breaking security is at least as hard as breaking a hard problem H. In this instance we say we have reduced the security to H. We comment now on the concept of *hard problems* and how they relate to our work.

Hard problems — for example the so called *factoring problem* or *discrete log problem* — lie at the heart of modern cryptography. In particular, hard problems form the basis of the *one way functions* that cryptographic constructions are often based on. The problem then is simple; if one can be certain that the problem is indeed *hard* and therefore that the underlying function is indeed *one way* then we have *security*. In fact much of the cryptographic research is focused on this and many would argue the equivalence of such a proof, showing the invertability of the underlying *one way function*, to a *proof of security*. However, many of the weaknesses of cryptographic constructions, in the real world, have not been on the underlying *one way function* but on the surrounding construction — *the protocol*. It is this latter question concerning the security of *the protocol* that we are most interested in here. We do need to reason about the underlying *one way function* or hard problem, but we do not consider how hard it is to invert — we only consider a reduction to it.

In this work we take inspiration from the improvements made to the way paper proofs are structured (as discussed in Section 1.1), along with the call for the use of formal methods made by Halevi and consider the formalisation of Multi-Party Computation (MPC), Σ-protocols and Commitment Schemes all of which are fundamental to modern cryptography.

MPC aims to provide protocols for mutually distrusting parties who wish to jointly compute functions over their inputs while keeping their inputs private. Work on MPC can be traced to Yao [74] where he posed and proposed the first solution to the problem. Initially MPC was considered an intellectual curiosity among cryptographers. However, advances in the last decade and improvements in efficiency and increased demand due to data protection regulations and industry needs mean it is now starting to be deployed in the real world. For example, it has been used for auctioning [19], secure email filtering and teleconferencing [47] and private statistical computation, e.g., using Sharemind [18]. It is this potential large scale implementation of MPC that heightens the need to examine it under the lens of formal verification.

Commitment Schemes and Σ-protocols are both two party primitives that are fundamental to modern cryptography. Commitment schemes allow for a party (the committer) to commit to a chosen message without revealing what the message is (the hiding property) and then reveal it at a later time in such a way that the other party is convinced that the committer has not changed the message that was committed to (the binding property).

Σ-protocols, on the other hand, are run between a Prover and a Verifier. The idea is that the Prover is able to convince a verifier that they know a value x without revealing anything beyond the fact they know x. They provide a baseline for Zero-Knowledge Proofs as they provide the Zero-Knowledge property (the Verifier learns nothing except that the Prover knows x) if the Verifier is honest.

Both Commitment Schemes and Σ-protocols are used extensively as building blocks for larger protocols. In particular, they are often used in MPC protocols, allowing the protocols to prevent parties from cheating by holding them to account. In fact both these primitives are used in converting MPC protocols secure in the semi-honest model (a weaker adversary model) to protocols that are secure in the malicious model (a stronger adversary model). It is in this way we see our study of MPC, and Commitment Schemes and Σ-protocols being used in the future. We discuss this in more detail in the Conclusion in Section 9.1.3.

1.3 theorem proving and isabelle/hol

In layman's terms *theorem proving* is the process of getting computers to prove theorems, either automatically, or by the human interacting with the machine and *coding* a proof. More precisely theorem proving allows the proof of mathematical statements, that are stated using a formal logic, using a *proof language*. Proofs resulting from theorem proving come with a degree of correctness far beyond that of a traditional paper proof.

In this work we use the proof assistant Isabelle/HOL[1] [59]. Isabelle/HOL is written in Standard ML and provides an implementation of Higher-Order-Logic (HOL). At the heart of Isabelle is a small trusted kernel [40] that implements the HOL proof axioms (rules) and the simply typed λ-calculus. These proof rules are used to decide if a term should be a theorem or not — a theorem is only accepted if it has been constructed using the proof rules. In this way all theorems are reduced to the small trusted kernel thus it is only implementation errors in the kernel (a few thousand lines of code that has been thoroughly scrutinized) that could result in false theorems being accepted.

Having, hopefully, convinced the reader of the robustness of theorem proving in Isabelle we move to highlight three reasons we believe Isabelle is a suitable and strong environment to work in. The first is the level of automation available. The sledgehammer tool [13] reduces the workload for the whole spectrum of Isabelle users: it points beginners to theorems they did not know existed, and often surprises experienced users by the proofs it can provide. To quote Paulson et al. [62] Sledgehammer "is now seen as indispensable".

The second aspect of Isabelle we highlight is the Isar proof language and in particular Wenzel's Isar language of structured proofs [73]. A common issue with proof assistants is that the proof scripts produced are incomprehensible to any human other than the author (and as time goes on they usually become incomprehensible to the author too!). Again to quote Paulson et al. [62]:

> 'The problem with these traditional approaches is that somebody looking at a machine proof can have no idea what is being proved at a given point: it is like playing blindfold chess.'

The idea behind Isar is to hierarchically structure the proof and prove *local* goals using *local* assumptions. In essence the proof can be split up into parts and consequently become considerably more readable and efficient. For example one can use the Isar language to prove an *iff* statement as follows [58]:

```
show $P \longleftrightarrow Q$
proof
  assume $P$
  $\vdots$
  show $Q$ ⟨proof⟩
next
  assume $Q$
  $\vdots$
  show $P$ ⟨proof⟩
qed
```

Due to the nature of our proofs — proving relationships between probabilistic programs — we find the Isar proof structure very useful. For example, if we wanted

[1] Throughout we refer to Isabelle/HOL or Isabelle, in both cases we mean the same thing.

to prove that the probabilistic programs A_1 and A_n are equal the proof would be structured as follows, where $A_2, \dots, A_{n-1}$ are intermediate probabilistic programs we define and use in the proof. The Isar proof allows us to chain our reasoning in a neat, readable way. In particular, even if the proofs of the validity of the individual steps in the overall proof are not accessible to the non-expert, the effect of each step on the original program A_1 can be seen easily.

```
show $A_1 = A_n$
proof
  have $A_1 = A_2$ $\langle proof \rangle$
  also have $\dots = A_3$ $\langle proof \rangle$
  also have $\dots = A_4$ $\langle proof \rangle$
  $\vdots$
  also have $\dots = A_{n-1}$ $\langle proof \rangle$
  ultimately show thesis $\langle proof \rangle$
qed
```

Finally we highlight the Isabelle distribution [45] and the Archive of Formal Proofs (AFP) [3]. The distribution comes equipped with about 750,000 lines of Isabelle/HOL theory covering basic (and not so basic) mathematics and computer science. This means one can pick up Isabelle and use many lemmas and theorems out the box. The automation helps here as finding the useful statements from the distribution can often be challenging, even with the search methods Isabelle offers. Complementing the Isabelle/HOL distribution is the AFP. The AFP comprises of a large collection of proof theories that have been developed by the Isabelle community. It is common that for a conference or journal publication, where the formalisation has been completed in Isabelle, a corresponding AFP entry will also be published. Submissions are reviewed before entry to the AFP. Statistics regarding the AFP are available online [3], at the time of writing (February 2020) there are: 518 entries from 342 authors with over 141,000 lemmas and nearly 2,500,000 lines of code. One main benefit of the AFP is that it is not a static environment. For each Isabelle release there is a corresponding AFP release, meaning that all AFP entries work with the current Isabelle release making it easy to build on work that is in the AFP.

All the work presented in this thesis is available in the AFP [20, 24].

1.4 OUTLINE OF THESIS

Here we outline the contents of each chapter. After presenting the preliminary material we split the main body of the thesis into two parts, the first concerning Σ-protocols and Commitment Schemes and the second concerning MPC.

At the end of the section we depict our contributions in Figures 1.1 and 1.2.

CHAPTER 2: Σ-PROTOCOLS, COMMITMENT SCHEMES AND MPC

Here we introduce the relevant security definitions and primitives we consider throughout the rest of the thesis. Specifically we introduce the simulation-based definitions of security for MPC and the game based definitions for Σ-protocols and commitment schemes — the definitions for Σ-protocols have a flavour of simulation too. Throughout the section we provide example protocols and detailed low level proofs.

CHAPTER 3: ISABELLE AND CRYPTHOL

In this chapter we introduce Isabelle/HOL and CryptHOL. In introducing CryptHOL we focus on the parts we use heavily in our work. We also outline our workflow from formalising a set of security properties to instantiating them for protocols in CryptHOL and discuss how we deal with the security parameter in our work.

PART I: FORMALISING Σ-PROTOCOLS AND COMMITMENTS

CHAPTER 4: Σ-PROTOCOLS

In this chapter we first show how we formalise Σ-protocols before considering multiple instantiations. We consider compound AND and OR statements for Σ-protocols and then the Σ-protocols due to Schnorr [68], Chaum-Pedersen [27] and Okamoto [61]. The AND and OR constructions allow us to leverage the module system in Isabelle to produce proofs at a general level, assuming only that the underlying Σ-protocols exist. Our work here allows us to highlight the correct definition of Σ-protocols from the many in the literature. We take time to discuss the different definitions of Σ-protocols given in the literature and how we are able to select the correct one.

CHAPTER 5: COMMITMENT SCHEMES

Here we formalise a framework to reason about the security of commitment schemes and show how it can be instantiated for the Rivest commitment scheme. The Rivest scheme uses a trusted initialiser to distribute co-related randomness to each party and thus is not of the form of a standard commitment scheme. We also prove the Pedersen commitment scheme secure however this is done from the general proof of the construction of commitment schemes from Σ-protocols in Chapter 6. We do, however, provide a proof of the Pedersen commitment scheme from scratch in our formalisation to compare the difference in proof effort; the proof from scratch consists of about 500 lines of proof whereas the proof we detail in the Chapter 6 comes in about 20 lines of proof.

CHAPTER 6: COMMITMENT SCHEMES FROM Σ-PROTOCOLS

We combine our work in the previous two chapters to formalise the construction of commitment schemes from Σ-protocols. We provide the proof of the construction at an abstract level (like in the AND and OR constructions of Σ-protocols), assuming only that the underlying Σ-protocol exists. Using this we instantiate for the Σ-protocols we consider, namely the Schnorr, Chaum-Pedersen and Okamoto protocols. In this thesis we focus on the instantiation of the Schnorr Σ-protocol with the general proof as this yields the Pedersen commitment scheme.

PART II: FORMALISING MULTI-PARTY COMPUTATION

CHAPTER 7: SEMI-HONEST SECURITY

This chapter first details our formalised framework for semi-honest security in the two party setting. We instantiate our framework for various MPC protocols. First we consider 1-out-of-2-OT (OT^1_2): we formalise an OT^1_2 constructed from a general Extended Trapdoor Permutation (ETP) [48] and then show how this proof can be instantiated for the RSA collection of functions (a known ETP). Second we consider the Naor-Pinkas OT^1_2 [57], a protocol that is widely used in the real world. Next, we build on this work in a modular way to show the two party GMW [39] protocol secure. Finally, we consider a secure multiplication protocol. This protocol uses a trusted initialiser, showing some of the flexibility of our framework.

CHAPTER 8: MALICIOUS SECURITY

Here we detail our formalised framework to consider malicious security in the two party setting. We then instantiate our framework to prove an OT^1_2 protocol [44] secure. The proof shows how the malicious setting is considerably more complex than the semi-honest setting.

CHAPTER 9: CONCLUSION

We summarise our work in this thesis and discuss its limitations. We also consider related work and discuss how our work fits into the literature.

CONTRIBUTIONS We summarise the contents and contribution of this thesis in the following diagrams, Figures 1.1 and 1.2. Figure 1.1 outlines for our formalisation on Commitment Schemes and Σ-protocols and Figure 1.2 details our formalisations for MPC. We reproduce these diagrams in part in the relevant chapters of the thesis. In the diagrams solid arrows represent proofs of concrete protocols; the arrows end at the instantiated framework. The double arrow (Figure 1.1) represents our formalisation of the general construction of commitment schemes from Σ-protocols, and the corresponding commitment schemes from our instantiated Σ-protocols whose security statements come for free due to the general proof. We highlight one of these in particular in the Figure with the dotted arrow as the instantiation of the Schnorr Σ-protocol under the general construction yields the well known Pedersen commitment scheme — the other instantiated proofs for the Σ-protocols we consider are captured by the double arrow representing the general proof and are not shown explicitly.

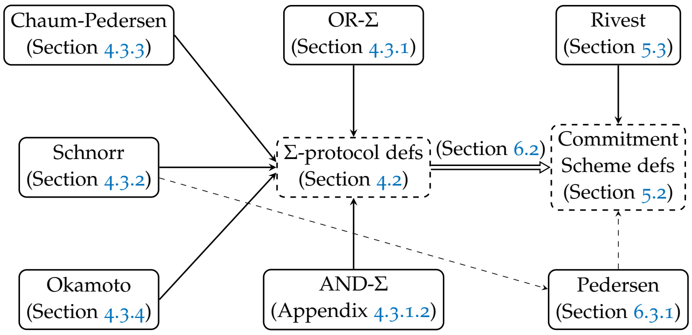


Figure 1.1: An outline of the formalisation of Σ-protocols and Commitment Schemes in this thesis.

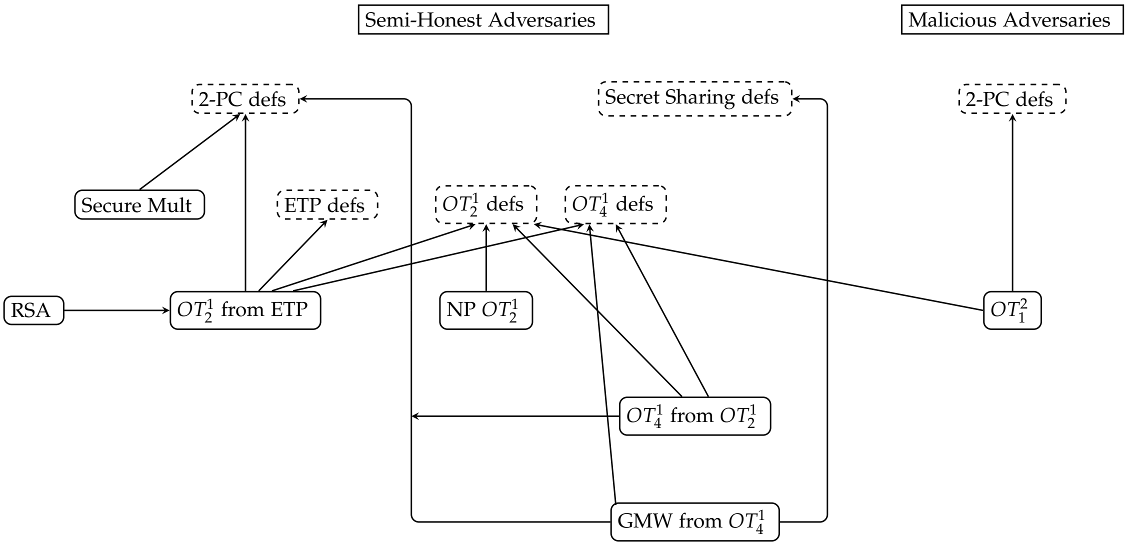


Figure 1.2: An outline of the formalisation of MPC in this thesis.

2 Σ-PROTOCOLS, COMMITMENT SCHEMES AND MPC

In this chapter we introduce the three areas of cryptography we study in this thesis. We bias the presentation of material to that which is relevant for our work; this is not a complete introduction to the three areas.

We provide the definitions we require for the rest of this thesis as well as examples of protocols that realise the definitions. Also, in the case of Σ-protocols and commitment schemes we show the general constructions we consider later in the thesis — namely compound Σ-protocols and the construction of commitment schemes from Σ-protocols. We provide detailed proofs of all examples and constructions in this chapter to allow the reader to gain an in depth understanding of why the definitions are met. For completeness, we reintroduce these constructions and some of the examples briefly later in the thesis when we formalise them. This section, however, is intended as the main reference for the rest of the thesis.

We make one comment regarding the security parameter. The security parameter provides a measure of how *hard* it is to break a cryptographic scheme, and can be increased and decreased depending on the required level of security. Typically the security parameter determines the *hardness* of the underlying computational problem of the scheme or protocol, for example if the security relied on the discrete log problem (considered over a cyclic group) then the security parameter would determine the size of the cyclic group over which the protocol was run. Often however the security parameter is assumed to be implicit in the definitions, depending on the particular paper definition. When introducing MPC we include the security parameter explicitly, this is because nearly every modern presentation of the definitions does this [37, 48]. On the other hand, definitions for Σ-protocols and commitment schemes often do not explicitly present the security parameter, thus, here we do not rely on it to match the paper proofs. We explain how we deal with the security parameter in our formalisation in Section 3.3.3.

We formalise all the definitions and examples given here, along with more case studies in the remaining parts of this thesis.

2.1 PRELIMINARIES

In this section we introduce some notions we rely on in the rest of the chapter.

2.1.1 *Game-based proofs*

Security games are used to "tame complexity" [71] of security proofs. Our main use for security games are in defining security for commitment schemes. In this case the security games we consider can be considered as pseudo protocols played between the Committer and the Verifier. One of the parties is controlled by an adversary and the other is the challenger. Consider the hiding game (depicted later in Figure 2.3).

In this instance the Committer is the challenger and the Verifier the adversary; the keys are distributed and the adversary asked to output two messages of its choosing. The Committer picks one of the messages at random (flips a coin with outcome b) and constructs its commitment. The adversary is required to output its guess (b') as to which message was committed and wins the game if it guesses correctly. More generally the definition of security with respect to a security game is tied to an event E (in the hiding game this is $b = b'$). Security requires that the probability that E occurs *close* to some target probability (this is $\frac{1}{2}$ for the hiding property). The difference between the probability of the event E occurring and target probability is called the *advantage* of the adversary. Intuitively, security is achieved if this advantage is small.

The game-based approach allows the cryptographer to be more formal in their reasoning about security properties. In particular games afford the opportunity to provide rigorous proofs of security. A proof is generally structured as follows: let $G_0, \dots, G_n$ be a sequence of games where G_0 is the original security game and G_n is a game where the target probability is met. In the proof one shows that $|Pr[G_i] - Pr[G_{i+1}]|$ is small and thus the value of $|Pr[G_0] - Pr[G_n]|$ is also small. This is sometimes called the "game hopping" technique.

2.1.2 *Simulation-based proofs*

Simulation-based proofs consider the real-ideal world paradigm. This method is used to define security for MPC. The ideal world is constructed to be secure by definition. For example the ideal world may employ a trusted party to do some computation. To define security the real world must be indistinguishable from the ideal world.

To reason about indistinguishability the concept of computational indistinguishablity is used. We take the definition from [49], adapting it only for our input type. We note that in [49] Lindell takes inputs of type bitstring, we have inputs of arbitrary types.

Definition (informal) 1. *A probability ensemble $X = \{X(a, n)\}$ is an infinite sequence of random variables indexed by input a and security parameter n. X and Y are said to be computationally indistinguishable, $X \overset{c}{\equiv} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function μ such that for every a and n we have,*

$$|Pr[D(X(a, n)) = 1] - Pr[D(Y(a, n)) = 1]| \leq \mu(n).$$

A function is negligible if it approaches 0 faster than any inverse polynomial as n grows.

In general the simulation-based method allows security properties of protocols to be captured, whereas the game-based method is used to capture the security of primitives with games designed for each security property.

2.1.3 *One time pad*

Many of the properties we prove rely on the idea of a one time pad. A one time pad provides *perfect security* in the sense that it totally masks a secret value.

Consider a cyclic group G with generator g and a secret value $c \in G$. If we sample $x \xleftarrow{\$} \mathbb{Z}_{|G|}$, then the distributions on $c \cdot g^x$ and g^x are equal. That is an adversary will learn nothing about c by seeing $c \cdot g^x$, assuming the randomness x is not used elsewhere. Here x is the one time pad.

2.2 Σ-PROTOCOLS

Σ-protocols are a two party primitive run between a Prover and a Verifier. They allow the Prover to convince the Verifier they have some knowledge, x, without leaking anything information about x. They provide a baseline for Zero-Knowledge protocols as they require the Verifier to be honest.

In 1996 Cramer [29] introduced the abstract notion of a Σ-protocol, coined the term Σ-protocol, and gave the definitions of the properties we consider here. He also developed a rich theory of Σ-protocols that goes beyond what we formalise in this work. The idea, however, of a Σ-protocol was conceived before this. The first efficient Σ-protocol was introduced by Schnorr [68] in 1989 — we give this protocol in Example 1 and formalise it in Section 4.3.2.2.

Σ-protocols are a fundamental two party primitive that allow a Prover to convince a Verifier that they know a value x without revealing anything beyond the fact they know x. They could be considered the precursor or baseline of Zero Knowledge proofs — Σ-protocols however provide weaker security guarantees as the Verifier is required to act honestly. The consequence of this limitation is that Σ-protocols are usually far more efficient than Zero Knowledge proofs. An analogy can be drawn between this setting and the simulation based setting we will see in section 2.4: Σ-protocols lie in the semi-honest setting as we require the Verifier to follow the protocol whereas Zero Knowledge lies in the malicious setting as the Verifier is allowed to behave arbitrarily.

In introducing Σ-protocols and their properties we follow Damgård [31], Hazay and Lindell [44] and Cramer [29].[1]

A Σ-protocol is considered with respect to a relation R. If $(h, w) \in R$ then h is considered an instance of a computational problem where w is the witness or solution to the problem. For example consider the discrete log relation which is considered over a group G with generator g. We say w is a witness to $h \in G$ if the following relation holds.

$$(h, w) \in R_{DL} \iff h = g^w \tag{2.1}$$

The discrete log relation is widely used in cryptography as for certain groups (e.g $\mathbb{Z}_p^*$ and elliptic curves over finite fields) it is considered a hard relation, that is, it is computationally infeasible to obtain the witness w from h.

Any relation, R, gives rise to a language $L_R = \{h.\ \exists w.\ (h, w) \in R\}$ that consists of statements in R. That is, for a statement h to be in L_R there must exist a witness w such that $(h, w) \in R$.

[1] It turns out that Damgård [31] and Hazay and Lindell's definitions[44] are too weak. Our definition of a Σ protocol in Definition 3 therefore includes Cramer's additional requirements. A detailed discussion can be found in section 4.2.1.

A Σ-protocol is a three move protocol run between a Prover (P) and a Verifier (V) where h is common public input to both P and V and w is a private input to P such that $(h, w) \in R$.

Definition (informal) 2. *A Σ-protocol has the following three part form:*

$$\begin{array}{ccc} \textit{Prover} & & \textit{Verifier} \\ (a, r) \leftarrow \textit{initial} & \xrightarrow{\qquad a \qquad} & a \\ e & \xleftarrow{\qquad e \qquad} & e \leftarrow \textit{challenge} \\ z \leftarrow \textit{response} & \xrightarrow{\qquad z \qquad} & z \\ & & \textit{accepts / rejects} \end{array}$$

That is: first the Prover sends an initial message a which is created using randomness r (sampled by the Prover), second the Verifier sends a challenge e and finally the Prover sends a response, from which the Verifier decides if it will accept or reject the proof.

A *conversation* for an execution of a Σ-protocol is the transcript of the protocol — the tuple (a, e, z). The conversation is said to be *accepting* if the tuple corresponds to the outputs of the three moves in the protocol and the Verifier accepts the response z.

There are three properties that are required for a protocol of the above form to be a Σ-protocol. The definition we provide is an extended version of Damgård [31] and Hazay and Lindell's definitions[44], the extension is in the HVZK property (Condition 2). This property is required by Cramer [29], but as far as we know is only ever described in informal text in his work.

Definition (informal) 3. *Assume a protocol, π, of the above form is run between P and V. Then π is a Σ-protocol for a relation R if the following properties hold:*

- *Completeness: if P and V follow the protocol on public input h and private input w such that $(h, w) \in R$, then V always accepts.*
- *Special soundness: there exists a polynomial time adversary, $\mathcal{A}$, such that when given a pair of accepting conversations (on public input h) (a, e, z) and (a, e', z') where $e \neq e'$ it can output w such that $(h, w) \in R$.*
- *Honest Verifier Zero-Knowledge (HVZK): The following conditions must hold.*
  1. *There exists a polynomial-time simulator S that on input h (public input) and e (a challenge) outputs an accepting conversation (a, e, z) with the same probability distribution as the real conversations between P and V on input (h, w). That is for all h and w such that $(h, w) \in R$ and every e we have*

$$\{S(h, e)\} = \{\langle P(h, w), V(h, e)\rangle\}$$

     where $\{S(h, e)\}$ is the output distribution of the simulator and $\{\langle P(h, w), V(h, e)\rangle\}$ denotes the distribution of the output transcript of an execution of the protocol between P and V.

2. *For $h \notin L_R$ the simulator $S(h,e)$ must nevertheless output an accepting conversation (a,e,z).*

Completeness provides a notion of correctness for the protocol, that is if the protocol is executed honestly then the Verifier will accept. The intuition for the special soundness property is that if a Prover can respond correctly to two different challenges then it can also compute the witness, meaning it is infeasible for a Prover to cheat a Verifier — that is, the Prover cannot convince the Verifier when a witness is not known. The HVZK property ensures that no information about the witness is leaked during the execution of the protocol. The first condition resembles definitions from Multi-Party Computation (MPC) where the real view (the real conversation generated by the Prover and Verifier) can be simulated without the private input (the witness). Condition 2 is not commonly included in literature definitions. We show is it required as it ensures the OR construction of Σ-protocols satisfies completeness. Condition 2 deals with the case where $h \notin L_R$, we note however there must still be a 'type' restriction on h, for example if the Σ-protocol requires h to be a group element then this restriction still applies — without it the definition would not be type correct. In our formalisation we ensure this by introducing a predicate *valid-input*, we do not introduce this here as sublties like this are rarely formulated in informal (paper) definitions.

Next we give an example of a Σ-protocol, namely the Schnorr Σ-protocol.

Example 1. *The Schnorr protocol uses a cyclic group G with generator g and considers the discrete log relation which on public input h requires the witness to be the discrete log of h in G so $h = g^w$. The protocol is run as follows.*

$$
\begin{array}{ccc}
\textit{Prover} & & \textit{Verifier} \\
(h,w) & & h \\
r \xleftarrow{\$} \mathbb{Z}_{|G|} & & \\
a \leftarrow g^r & \xrightarrow{\quad \textit{initial msg: } a \quad} & a \\
 & \xleftarrow{\quad \textit{challenge: } e \quad} & e \xleftarrow{\$} \mathbb{Z}_{|G|} \\
z \leftarrow (w \cdot e + r) \bmod |G| & \xrightarrow{\quad \textit{response: } z \quad} & z \\
 & & \textit{check: } a \cdot h^e = g^z \\
 & & \textit{accepts or rejects}
\end{array}
$$

Here we use $\xleftarrow{\$}$ to denote a uniform sampling and $\leftarrow$ to denote assignment.

It is easy to see the protocol has the required three phase form to satisfy Definition 2. The Prover holds (h,w) such that $h = g^w$ and the Verifier holds only h. The initial message sent by P to V is a uniformly sampled group element and the challenge is uniformly sampled from the field of size $|G|$ — it is important that the challenge is constructed honestly by the Verifier as this is an assumption on the security properties of Σ-protocols (the Verifier cannot act maliciously). The response is constructed by P as $z = (w \cdot e + r) \bmod |G|$ and sent to V who accepts or rejects based on whether $a \cdot h^e = g^z$.

Theorem 1. *The Schnorr protocol is a Σ-protocol.*

Proof. We must prove the three properties required for Σ-protocols: completeness, special soundness and HVZK.

COMPLETENESS Completeness can be seen by the following string of equalities: $a \cdot h^e = g^r \cdot (g^w)^e = g^{r+w \cdot e} = g^z$.

SPECIAL SOUNDNESS Given two accepting conversations (a, e, z) and (a, e', z') a witness can be extracted by taking $w = (\frac{z-z'}{e-e'})$ *mod* $|G|$. This can be seen as follows: we have $a \cdot h^e = g^z$ and $a \cdot h^{e'} = g^{z'}$, as both conversations are accepting. Rearranging this and using the fact $h = g^w$ we find $w = (\frac{z-z'}{e-e'})$ *mod* $|G|$. (We note it is important that the challenges e and e' are distinct, otherwise w is undefined — luckily distinct challenges is a requirement of the special soundness property.) Finally we must check the given adversary runs in polynomial time. This is clearly the case as it only performs elementary operations on the inputs it receives.

HONEST VERIFIER ZERO KNOWLEDGE We claim following the simulator is sufficient to show HVZK. The simulator is given public input h and challenge e as inputs and is constructed as follows:

1. $z \xleftarrow{\$} \mathbb{Z}_{|G|}$
2. $a \leftarrow g^z \cdot h^{-e}$
3. output (a, e, z)

We now consider the two parts of the HVZK definition:

1. We claim the output of the simulator is equivalent to the output of a real execution of the protocol. This is easily seen by observing that the real output is $\{g^r, e, g^{w \cdot e + r}\}$ and the simulated output is $\{g^{z - w \cdot e}, e, g^z\}$. If we let $r' = z - w \cdot e$ and substitute it into the simulated output we get $\{g^{r'}, e, g^{w \cdot e + r'}\}$, if z is uniformly sampled then so is r' hence the distributions are equal. Moreover the simulator clearly runs in polynomial time.
2. Recall for Condition 2 we still require $h \in G$. Using this we can realise the proof in two ways. First, if $h \notin L$ the simulator still outputs an accepting conversation. In particular we have $a \cdot h^e = g^z \cdot h^{-e} \cdot h^e = g^z$. Second, we note there is a contradiction if $h \in G$ and $h \notin L$ as G is a finite cyclic group, in particular no $h \notin L$ exists[2].

Thus we have shown the Schnorr Σ-protocol meets the three components of the definition of a Σ-protocol.

□

The intuition behind constructing the simulator for the HVZK property is to work backwards. We would like the response to leak no information about w, so let us pick it uniformly at random and then try to reconstruct the initial message. If we sample z uniformly from the field and then set $a = g^z \cdot h^{-e}$ we can show the resulting conversation gives a distribution equal to the output conversation distribution of a real execution of the protocol.

[2] This contradiction will arise for all discrete log based relations

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
(w_0,w_1),(x_0,x_1) & & (x_0,x_1) \\
(a_0,r_0) \leftarrow \mathit{initial}_0 & & \\
(a_1,r_1) \leftarrow \mathit{initial}_1 & \xrightarrow{\quad (a_0,a_1) \quad} & (a_0,a_1) \\
e & \xleftarrow{\quad e \quad} & e \leftarrow \mathit{challenge} \\
z_0 \leftarrow \mathit{response}_0 & & \\
z_1 \leftarrow \mathit{response}_1 & \xrightarrow{\quad (z_0,z_1) \quad} & (z_0,z_1) \\
 & & \mathit{checks}\ (a_0,e,z_0),(a_1,e,z_1)\ \mathit{are\ valid} \\
 & & \mathit{accepts/rejects}
\end{array}$$

Figure 2.1: The AND construction for Σ-protocols.

The HVZK property may appear to make Σ-protocols be too weak to be practically useful, however it turns out that such protocols can be used as building blocks in constructions that are indeed secure against malicious Verifiers.

2.2.1 *Compound constructions for Σ-protocols*

The theory of Σ-protocols can be extended to allow for compound statements of multiple relations. In particular we consider OR and AND statements with respect to two Σ-protocols Σ_0 and Σ_1. The OR construction allows a Prover to prove that given two inputs x_0, x_1 they know w such that either $(x_0, w) \in R_0$ or $(x_1, w) \in R_1$ hold. Analogously the AND construction allows the Prover to prove that they know (w_0, w_1) such that both $(x_0, w) \in R_0$ and $(x_1, w) \in R_1$ hold. We show both the constructions next; the components of the respective underlying Σ-protocols are denoted with subscript 0 and 1 respectively (e.g., the algorithm that generates the initial messages and randomness for the Σ-protocol for R_0 is denoted as $\mathit{initial}_0$)

2.2.1.1 *The AND construction*

The AND construction is the simpler of the two constructions. The relation we consider is as follows:

$$((x_0, x_1), (w_0, w_1)) \in R_{AND} \iff (x_0, w_0) \in R_0 \wedge (x_1, w_1) \in R_1 \tag{2.2}$$

The intuition is that the two Σ-protocols are run in parallel; if the Prover knows the witness for both public inputs then they are able to respond correctly to challenges with respect to both underlying Σ-protocols. We sketch the protocol below in Figure 2.1.[3]

We highlight that the *challenge* must be compatible for both protocols. In practice it would be a bitstring, in our formalisation we generalise to run the protocol over

[3] We say sketch as we do not provide the inputs to the various algorithms as we have not formally defined them yet. We introduce the protocol more formally in Figure 4.3.

any boolean algebra. We are able to make this generalisation as the only non trivial property we require in the proofs is the one time pad property, which we prove for a boolean algebra. This is the same for the OR construction.

Theorem 2. *The AND construction given in Figure 2.1 is a Σ-protocol.*

Proof. The protocol clearly has the correct three part form required in Definition 2. We consider the three properties we require from Definition 3 in turn.

COMPLETENESS The two conversations (a_0, e, z_0) and (a_1, e, z_1) are accepting conversations for their respective Σ-protocols as they are formed using the initial, challenge and response algorithms from the corresponding underlying Σ-protocols. Thus the proof of completeness comes from the underlying completeness properties.

SPECIAL SOUNDNESS We construct a special soundness adversary using the special soundness adversaries of the underlying Σ-protocols to output the respective witnesses.

HONEST VERIFIER ZERO KNOWLEDGE We can simulate the conversations by combining the simulators for the respective Σ-protocols. When the inputs are not in the respective languages we know the simulator still outputs an accepting conversation by the second HVZK property on the underlying simulators.

□

2.2.1.2 *The OR construction*

The OR construction is more complex. The relation we consider is as follows:

$$((x_0, x_1), w) \in R_{OR} \iff (x_0, w) \in R_0 \vee (x_1, w) \in R_1 \tag{2.3}$$

The intuition for the protocol is that the Prover is asked to prove complete two instances of the protocol, one for x_0 and one for x_1. For the x_i for which the Prover knows w the Prover can complete the Σ-protocol as in real life, and for the other case the Prover can simulate the protocol. The protocol is given in Figure 2.2. Here b denotes the Σ-protocol for which the Prover knows the witness.

Theorem 3. *The OR construction given in Figure 2.2 is a Σ-protocol.*

Proof. The protocol clearly has the correct three part form required in Definition 2. We consider the three properties we require from Definition 3 in turn.

COMPLETENESS The conversation generated by the simulator is valid by the HVZK property of R_{1-b}. We must use both properties of the HVZK property to cover the cases where $x_{1-b} \in L_{1-b}$ and $x_{1-b} \notin L_{1-b}$. In either case the conversation is accepting. The conversation generated by the real execution of the Σ-protocol for R_b produces an accepting conversation by the completeness property of the Σ-protocol for R_b.

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier}\\
(x_0,x_1),w & & (x_0,x_1)\\
(r_b,a_b) \xleftarrow{\$} \mathit{initial}_b & & \\
e_{1-b} \xleftarrow{\$} \mathit{challenge} & & \\
(a_{1-b},e_{1-b},z_{1-b}) \xleftarrow{\$} S_b(x_{1-b},e_{1-b}) & \xrightarrow{\text{initial msg: } (a_0,a_1)} & (a_0,a_1)\\
s & \xleftarrow{\text{challenge: } s} & s \xleftarrow{\$} \mathit{challenge}\\
e_b = s\oplus e_{1-b} & & \\
z_b \leftarrow \mathit{response}_b & \xrightarrow{\text{response: } (e_0,z_0,e_1,z_1)} & (e_0,z_0,e_1,z_1)\\
 & & \mathit{checks}\ s = e_0 \oplus e_1\ \mathit{and}\\
 & & (a_0,e_0,z_0),(a_0,e_0,z_0)\ \mathit{are\ valid}
\end{array}$$

Figure 2.2: The OR construction for Σ-protocols. Here b is the relation for which the witness holds. That is $(x_b, w) \in R_b$. S_b is the simulator for the HVZK property for Σ_b.

SPECIAL SOUNDNESS We construct the special soundness adversary to call the special soundness adversary of the Σ-protocol for which the witness is required. We know this exists by the special soundness property of the Σ-protocol for R_b.

HONEST VERIFIER ZERO KNOWLEDGE The real execution of the protocol can be simulated by running the simulator for both relations. The HVZK property follows from the HVZK property of R_b.

□

2.3 COMMITMENT SCHEMES

Commitment schemes were first introduced by Blum [14] and Even [34] in 1981. The problem Blum proposed was that of coin flipping by telephone; how do Alice and Bob flip a coin via telephone? Blum proposed commitments to solve such a problem: Alice *calls* the coin flip and commits to her call, Bob then flips the coin and reveals the result upon which Alice reveals the value she committed to so Bob can verify her call matches her commitment. If Alice's call matches the coin flip she wins. A commitment scheme, run between a Committer and Verifier, has the following structure.

Definition (informal) 4. *A commitment scheme has the following three part form:*

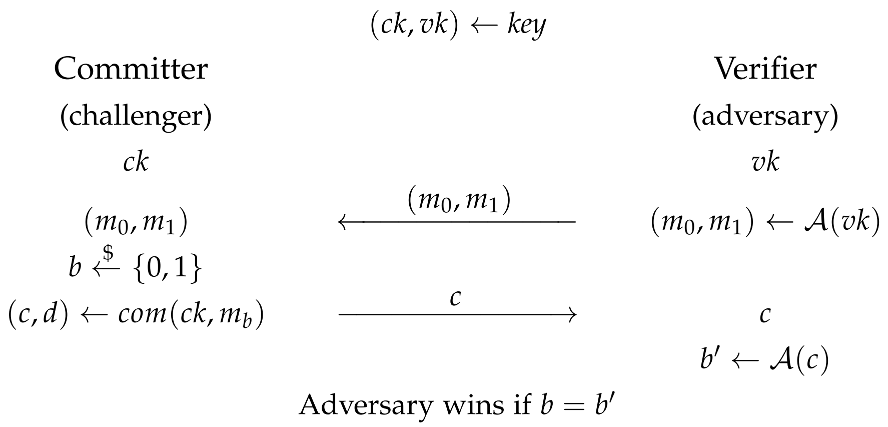
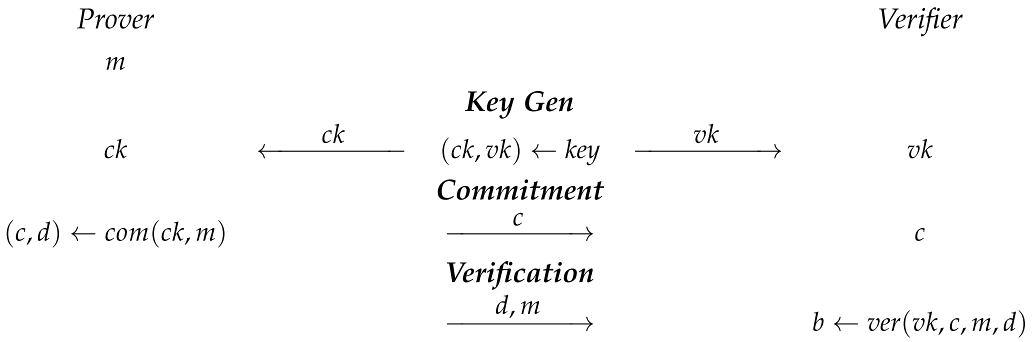

$(ck, vk) \leftarrow key$
Committer
(challenger)
ck
Verifier
(adversary)
vk
(m_0, m_1)
$(m_0, m_1) \leftarrow \mathcal{A}(vk)$
(m_0, m_1)
$b \xleftarrow{\$} \{0,1\}$
$(c, d) \leftarrow com(ck, m_b)$
c
c
$b' \leftarrow \mathcal{A}(c)$
Adversary wins if $b = b'$

Figure 2.3: The hiding game played between the Committer (the challenger) and the adversary (the Verifier).



That is: first the keys for each party are generated in the key generation phase. Often, like in the Pedersen commitment scheme $ck = vk$ thus they can be created by one of the parties and sent to the other. In the Rivest commitment scheme, however, each party receives private keys from a trusted initialiser. The above description (and our formalisation) allows for both cases. In the commitment phase the committer outputs the commitment c and an opening value d, which is sent to V in the verification phase. C sends c to V. Finally, in the verification phase the verifier takes the verification key, commitment, original message and opening value as input and outputs a boolean depending on whether the verification is successful.

The three properties we want from a commitment scheme are correctness, hiding and binding.

Definition (informal) 5 (Correctness)**.** *A commitment scheme is said to be correct if whenever the protocol is run honestly between C and V, then V will always accept in the verification phase for all messages that can be committed.*

To define the hiding and binding properties, cryptographers consider two security games that are played between an adversary and a benign challenger. Games are used to *tame complexity* [71] of security proofs. The security games we consider can be considered as pseudo protocols played between the Committer and the Verifier, where one of the parties is controlled by an adversary and the other is the challenger. Consider the hiding game depicted in Figure 2.3. Here the Committer is the challenger and the Verifier the adversary; the keys are distributed and the adversary asked to output two messages of its choosing. The Committer picks one of the messages at random and constructs its commitment. The adversary is required to output its guess as to which message was committed and wins the game if it guesses correctly. More generally the definition of security with respect to a security game is tied to an event E (in the hiding game this is $b = b'$), security requires that the probability that E occurs *close* to some target probability (this is $\frac{1}{2}$ for the hiding property) — the difference between the probability of the event E occurring and target probability is called the advantage of the adversary. Intuitively security is achieved if this advantage is small.

To define the hiding property we consider the algorithm which plays out the hiding game from Figure 2.3. Informally the algorithm, *hid-game*, is as follows where $\mathcal{A}$ is the adversary playing against the challenger:

1. $(ck, vk) \leftarrow key$
2. $(m_0, m_1) \leftarrow \mathcal{A}(vk)$
3. $b \overset{\$}{\leftarrow} \{0, 1\}$
4. $(c, d) \leftarrow com(ck, m_b)$
5. $b' \leftarrow \mathcal{A}(c)$
6. $return\ b = b'$

Definition (informal) 6 (Hiding)**.** *The hiding advantage is defined for all polynomial-time adversaries, $\mathcal{A}$, as*

$$hid\text{-}adv(\mathcal{A}) = |Pr[hid\text{-}game(\mathcal{A}) = 1] - \frac{1}{2}|$$

The scheme is said to be perfectly hiding if for all adversaries, $\mathcal{A}$, we have

$$hid\text{-}adv(\mathcal{A}) = 0.$$

The scheme is said to be computationally hiding if for all computationally bounded adversaries, $\mathcal{A}$, the advantage value $hid\text{-}adv(\mathcal{A})$ is negligible. [4]

Analogously to the hiding property we define the binding property with respect to the binding game which is depicted in Figure 2.4. The informal algorithm for playing the binding game (*bind-game*) with adversary $\mathcal{A}$, is as follows:

1. $(ck, vk) \leftarrow key$
2. $(c, m, d, m', d') \leftarrow \mathcal{A}(ck)$
3. $checks\ m \neq m'$
4. $b \leftarrow ver(vk, c, m, d)$

[4] Computational bounds and negligibility are typically used in asymptotic security statements. There, all definitions are parametrised by a security parameter η and an adversary's run-time must be bounded by a (polynomial) function of η. Then, the advantage is negligible if it approaches 0 faster than any inverse polynomial as the security parameter grows.

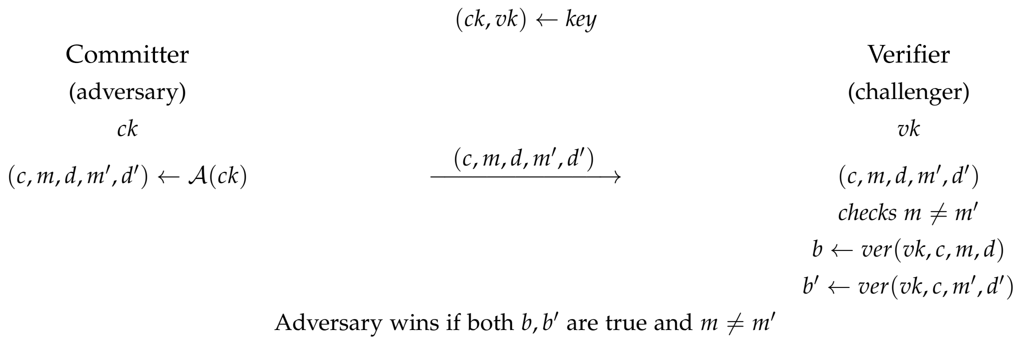

$(ck, vk) \leftarrow key$
Committer
(adversary)
ck
$(c, m, d, m', d') \leftarrow \mathcal{A}(ck)$
$\xrightarrow{(c, m, d, m', d')}$
Verifier
(challenger)
vk
(c, m, d, m', d')
$checks\ m \neq m'$
$b \leftarrow ver(vk, c, m, d)$
$b' \leftarrow ver(vk, c, m', d')$
Adversary wins if both b, b' are true and $m \neq m'$

Figure 2.4: The binding game played between the challenger (the Verifier) and the adversary (the Committer).

5. $b' \leftarrow ver(vk, c, m', d')$
6. *return* $b' \wedge b$

Intuitively the challenger asks the adversary to output two messages (m, m') and corresponding opening values (d, d') for the same commitment c. If the adversary can achieve this such that both messages (and corresponding opening values) verify then the adversary (the Committer) is not *bound* to the original message they commit to.

Definition (informal) 7 (Binding)**.** *The binding advantage is defined for all polynomial-time adversaries, $\mathcal{A}$, as*

$$bind\text{-}adv(\mathcal{A}) = Pr[bind\text{-}game(\mathcal{A}) = 1]$$

The scheme is said to be perfectly binding if for all adversaries, $\mathcal{A}$, we have

$$bind\text{-}adv(\mathcal{A}) = 0.$$

The scheme is said to be computationally binding if for all computationally bounded adversaries, $\mathcal{A}$, the advantage bind-adv$(\mathcal{A})$ is negligible.

We revert back to our coin flipping example to give some intuition regarding these properties. In the example Alice is the Committer and Bob the Verifier. Firstly we want the scheme to be correct, that is if both parties run the commitment protocol in the prescribed way then the Verifier will always be convinced in the verification phase. Secondly, we do not want Bob to be able to learn anything about Alice's call (what she commits to) from the commitment itself — we want the commitment to be hiding. Finally we do not want Alice to be able to decommit to a different call of the coin flip from the one she committed to, that is we want her commitment to be binding.

We next give an example of a commitment scheme, the Pedersen commitment scheme [63] provided by Pedersen in 1991.

Example 2. *The Pedersen commitment scheme is run using a cyclic group of prime order G with generator g.*

$$pk \xleftarrow{\$} G$$

$$\begin{array}{ccc}
\textit{Committer} & & \textit{Verifier} \\
 & & \\
 & \textbf{\textit{Commitment}} & \\
d \xleftarrow{\$} \mathbb{Z}_q & & \\
c \leftarrow g^d \cdot pk^m & \xrightarrow{\quad c \quad} & c \\
 & \textbf{\textit{Verification}} & \\
 & \xrightarrow{\quad (m,d) \quad} & (m,d) \\
 & & \textit{checks } c = g^d \cdot pk^m \\
 & & \textit{accepts / rejects}
\end{array}$$

It can be easily seen the protocol has the desired form required in Definition 4. We next prove the protocol meets the definitions of a commitment scheme.

Theorem 4. *The Pedersen scheme given above is a commitment scheme.*

Proof. We address the three properties: correctness, hiding and binding in turn.

CORRECTNESS If the protocol is run honestly the Verifier will always accept in the verification phase. This is seen trivially as the Verifier explicitly checks the value of c.

HIDING We show perfect hiding for the Pedersen commitment scheme. An adversary playing the hiding game receives the commitment, $c = g^d \cdot pk^{m_b}$, of one of the two messages they choose (m_0, m_1). We show the commitment they receive reveals no information about the message m_b. This is seen as the distribution over the commitment is the same as returning a uniform sample from the group. That is, the distribution over $g^d \cdot pk^{m_b}$ is equal to the distribution over g^d when d is uniformly sampled — this is due to the one time pad property.[5] Therefore the hiding game is equivalent to the adversary receiving back g^d and having to guess which message was used to make the commitment. Clearly as g^d does not depend on m_b the adversary must guess, giving them a $\frac{1}{2}$ chance of guessing correctly and winning the game. Consequently perfect hiding holds.

BINDING To prove the protocol is binding we show a reduction to the discrete log problem. In particular, we show if an adversary can win the binding game then we can construct an efficient adversary that beats the discrete log problem — it is able to find the discrete log of pk. Let (c, m, d, m', d') be the output from the adversary in the binding game. Let us also assume both messages and opening values correspond valid commitments to c, that is $c = g^d \cdot pk^m$ and $c = g^{d'} \cdot pk^{m'}$. Using this we have $pk = g^{\frac{d-d'}{m-m'}}$. This is well defined as we know $m \neq m'$ as it is a requirement on the output of the adversary in the binding game. Therefore we have shown we can construct an efficient adversary that can output the discrete log of pk — $\frac{d-d'}{m-m'}$ — which is assumed to be a hard problem, thus our reduction holds. □

[5] In our formal proof we must prove this is in fact a one time pad, however here we do not provide a proof as it is considered a well known result.

2.3.1 *Commitments from Σ-protocols*

Σ-protocols and commitment schemes are strongly related. Damgård [31] showed how Σ-protocols can be used to construct commitment schemes that are perfectly hiding and computationally binding and thus showed how these two fundamental cryptographic primitives are linked. The construction works as follows.

Consider a relation R for a Σ-protocol such that *gen* generates h and w such that $R(h,w)$ is satisfied. Using a Σ-protocol for the relation R we can construct the commitment scheme given below. The only restriction is the committer must commit to a message m that is a possible challenge. We denote this space by the set *challenge-space*. In reality this does not pose any restriction on m than any standard commitment scheme where, for example, we may require m to be in the group G, as in the Pedersen commitment scheme. In the key generation phase the Verifier runs the generation algorithm, $(h,w) \leftarrow gen$ and sends h to the Committer. To commit to a message e the Committer runs the simulator on their key h and e; that is they run $(a,e',z) \leftarrow S(h,e)$ and then send a to the Verifier and keep e' and z as the opening values. In the verification stage the Prover sends e' and z to the Verifier who uses the check algorithm of the Σ-protocol to confirm that (a,e',z) is an accepting conversation, with respect to the public input h. The protocol is seen in the diagram below.

$$\begin{tikzcd}[column sep=huge]
\text{Committer} & & \text{Verifier} \\
e \in \textit{challenge-space} & & \\
 & \textbf{Key Gen} & \\
h & & (h,w) \overset{\$}{\leftarrow} \textit{gen} \arrow[ll, "h"'] \\
 & \textbf{Commitment} & \\
(a,e',z) \leftarrow S(h,e) \arrow[rr, "a"] & & a \\
 & \textbf{Verification} & \\
 \arrow[rr, "(e'{,}z)"] & & e',z \\
 & & \text{checks } (a,e',z) \\
 & & \text{accepts/rejects}
\end{tikzcd}$$

Theorem 5. *The scheme constructed from a Σ-protocol, shown above, is a commitment scheme.*

Proof. As usual we treat the three properties in turn.

correctness The protocol is correct as the simulator is equal to the real view when its inputs are in the relation, this is by the first condition of the HVZK property of Σ-protocols. We know the real view always outputs an accepting conversation by completeness of the Σ-protocol and thus the commitment scheme will always accept.

hiding For this construction we show perfect hiding. We must show the commitment, that is a outputted in $(a,e',z) \leftarrow S(h,e)$, is independent of the message being committed, e. By the HVZK property we know the simulated conversation is equal to a real conversation and in the real conversation we know that the initial message a (in this case the commitment) is generated independently of the challenge as it is constructed before the challenge. Thus the adversary in the hiding game must guess which message has been committed to based on a which is independent of the message (the challenge), therefore the adversary can at best win the hiding game with probability $\frac{1}{2}$.

BINDING The binding property follows from the special soundness property of the Σ-protocol; if the Committer could output the commitment a and opening values (e, z) and (e', z') such that both (a, e, z) and (a, e', z') are both accepting conversations then by the special soundness property there exists an adversary that can output the witness w which contradicts the assumption on the relation being hard. □

This construction allows for a new commitment scheme for every Σ-protocol that is proved secure. In fact under this construction the Schnorr Σ-protocol given in Example 1 gives a variant of the Pedersen commitment scheme given in Example 2. This construction was first shown by Damgård [32] in 1989.

2.4 MULTI-PARTY COMPUTATION

The goal of MPC is to allow mutually distrusting parties to compute a joint function of their inputs, while only revealing the output of the function and nothing of their respective inputs. Work on MPC can be traced to Yao [74] in 1981 where he posed and proposed the first solution to the problem. The problem posed by Yao was *the millionaires problem*: three millionaires are having lunch and they decide the richest among them will pay the bill but do not want to directly reveal their wealth to each other. Let us assume there are three millionaires: they want to compute the function $F_{wealth}(x, y, z) = max(x, y, z)$ while keeping their inputs to the function — their wealth — private. This example highlights an important point, even if they engage in MPC they will all learn something, namely who is the most wealthy as this is the output of the function. But they should learn no more. This scenario can be generalised for parties to have multiple inputs and potentially where different outputs get sent to different parties. In this work we only consider the two party setting.

MPC is realised by parties engaging in a protocol. When considering two party protocols the property we require of an MPC protocol is that of *input privacy*. Archer et al. [2] informally describes this:

> The information derived from the execution of the protocol should not allow any inference of the private data held by the parties, bar what is inherent from the output of the function.

The theoretic results on MPC are strong. Any function that can be represented as a boolean or arithmetic circuit can be computed under MPC. In general there are two techniques to achieve MPC, we describe them here in the two party setting.

The first, introduced by Yao in the oral presentation of his paper in 1986 [75], are so called *Garbled Circuits*. Here one party, the garbler, *garbles* (a randomised encoding) the circuit representing the function and encrypts their inputs and sends both the encrypted input and circuit to the other party, the evaluator. The evaluator then employs *oblivious transfer* to learn the encrypted values of their inputs and evaluates the circuit to obtain the encrypted outputs. The two parties communicate to decrypt the output.

The second technique is that of secret sharing and the *GMW* style protocols [39]. Here the parties first *share* their inputs with each other so they each hold part of each input — to share a bit x a party would randomly sample a bit a and compute $b = x \oplus a$ and send b to the other party. Then the parties use their shares to compute the gates in the circuit, after the computation of each, gate each party will hold a share of the output of the gate — the protocol will specify how the parties are to compute each gate using their respective shares. After computing the whole circuit together the parties will each hold a share of the output. They then reconstruct the output by combining their shares. To reconstruct the example sharing scheme earlier the two parties simply xor their shares, $x = a \oplus b$.

There are two standard adversary (corruption) models we consider in MPC. The first, weaker model, is the *semi-honest* model. Here we assume parties do not deviate from the protocol, that is they are *honest* in executing the protocol transcript. This may appear to be a weak adversary model however it provides a strong baseline for security. Protocols that are secure in the semi-honest model can often be extended to be secure in the malicious model. The semi-honest model is best employed when it is in the parties interest to participate honestly in the protocol. For example, consider two hospitals who want to compute a function over their patient records. The law likely states that they cannot pass the data to each other in the clear to compute the function as the data is sensitive, therefore the hospitals must take privacy preserving measures. In this situation MPC could be used, and in particular an MPC protocol that provides semi-honest security would suffice.

The second, stronger security model is the *malicious* model. Here parties are assumed to be fully corrupted and thus behave arbitrarily. This model is much stronger than the semi-honest model however requires more computation and overhead for the parties participating in the protocol. Even the malicious model has its limitations, for example it cannot stop a party refusing to participate in the protocol.

In the remaining sections of this chapter we show how semi-honest and malicious security is defined. We take our definitions from Lindell's tutorial [49] as it provides a concise and detailed account of simulation-based proofs in cryptography — it is considered a seminal tutorial in the area. We take much inspiration from Lindell's exposition of the definitions but take more time to emphasis parts of the definition that are particularly relevant to the formalisation.

2.4.1 *Defining Semi-Honest Security*

The semi-honest adversary model allows the adversary to control one of the parties but they must follow the protocol transcript exactly. Initially this may seem like a very weak security model — it does not capture even small deviations from the protocol description. It provides the guarantee that the protocol does not leak any information. For example, this model guarantees security against corruption of a party after the execution of the protocol. On a server on which the execution of the protocol took place, a later compromise cannot cause information leak beyond the server's own inputs.

FUNCTIONALITY The work here is constrained to two party protocols, such a protocol problem is defined by the notion of a *functionality*. A functionality is a map from inputs to outputs of each party. Let Party 1 and 2's inputs be of type $input_1$ and $input_2$ respectively and the respective outputs be of type $output_1$ and $output_2$. The functionality is denoted as

$$f \ : \ input_1 \times input_2 \rightarrow output_1 \times output_2. \tag{2.4}$$

This map is deconstructed to a pair of maps providing the output for each party, that is $f = (f_1, f_2)$ where $f(x,y) = (f_1(x,y), f_2(x,y))$. Here Party 1 obtains $f_1(x,y)$ and Party 2 obtains $f_2(x,y)$.

The functionality can be thought of as defining the problem that is solved by a protocol π. A protocol is the procedure the parties undertake together to finally obtain their desired outputs — the outputs specified by the functionality. Functionalities can be either deterministic, like Oblivious Transfer, or non-deterministic, like the secure multiplication we consider in section 7.5. Both are seen in the example below.

Example 3 (Functionalities)**.**

1. *A fundamental two party functionality is 1-out-of-2 Oblivious transfer (OT^1_2), it is defined as follows.*

   $$f_{OT^1_2}((m_0, m_1), \sigma) = (\_, m_\sigma)$$

   where _ denotes the empty string. Party 1 holds two messages and Party 2 a choice bit, the result of the functionality is that Party 2 learns its choice of message and Party 1 obtains nothing.

2. *The secure multiplication protocol we consider in section 7.5 allows two parties to compute the multiplication of their inputs, in fact each party obtains a* share *of the multiplication. The functionality, considered over the field $\mathbb{Z}_q$, is as follows,*

   $$f(x,y) = (s_1, x \cdot y - s_1)$$

   where $s_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$ is a uniform sample — this ensures neither of the outputs alone reveal the value of $x \cdot y$. The multiplication is obtained by adding the two parties' outputs, $x \cdot y = s_1 + (x \cdot y - s_1)$.

The functionality alone provides no information regarding the security of the protocol — in terms of guarantee's or requirements, it is solely a definition of the desired outcome of the protocol.

DEFINITION OF SECURITY Intuitively semi-honest security is realised if whatever can be computed by a party in the protocol can also be computed based on that party's input and output alone. To capture this notion the idea of simulation is used, security is realised if the real view of the protocol can be simulated in an ideal world where the private inputs are not used in its construction. To prove a protocol is secure against semi-honest adversaries for each party one must construct a simulator whose output is indistinguishable from the party's transcript in the real execution of the protocol. As the simulator for a party, by definition, is not allowed to see or use the private input(s) from the other party this implies that the parties can learn nothing from the execution of the protocol beyond what they can learn from their input and prescribed output — as determined by the functionality.

To define security the *views* of the parties are considered.

Definition (informal) 8. *Let π be a two party protocol with inputs (x, y) and with security parameter n.*

- *The real view of the i^{th} party (here $i \in \{1, 2\}$) is denoted by*

$$view_i^{\pi}(x, y, n) = (w, r^i, m_1^i, ..., m_t^i)$$

where $w \in \{x, y\}$ and is dependent on which view we are considering i.e. if $i = 1$ then $w = x$, r^i accumulates random values generated by the party during the execution of the protocol, and the m_j^i are the messages received by the party.

- *We denote the output of the i^{th} party, $output_i^{\pi}(x, y, n)$, and the joint output as*

$$output^{\pi}(x, y, n) = (output_1^{\pi}(x, y, n), output_2^{\pi}(x, y, n)).$$

Semi-honest security is defined as follows.

Definition (informal) 9 (Semi-honest security — non-deterministic functionalities). *A protocol π is said to securely compute f in the presence of a semi-honest adversary if there exist probabilistic polynomial time algorithms (simulators) S_1, S_2 such that*

$$\{S_1(1^n, x, f_1(x, y)), f(x, y)\} \overset{c}{\equiv} \{view_1^{\pi}(x, y, n), output^{\pi}(x, y, n)\}$$

$$\{S_2(1^n, y, f_2(x, y)), f(x, y)\} \overset{c}{\equiv} \{view_2^{\pi}(x, y, n), output^{\pi}(x, y, n)\}.$$

Sometimes it is possible to prove a stronger notion of security (usually only for one party in the protocol, it is known that for OT_2^1 we cannot have perfect security for both parties.), that is *perfect security*. For this to hold the two views are required to be equal, this trivially implies they are computationally indistinguishable also. Perfect security means that not even a computationally unbounded distinguisher can distinguish the views.

For a deterministic protocol it is shown in [49] that Definition 9 can be simplified. In particular we require a relaxed notion of security for each party along with the correctness property.

Definition (informal) 10. *[Semi-honest security — deterministic functionalities] For deterministic functionalities, f, π is said to securely compute f in the presence of a semi-honest adversary if there exist probabilistic polynomial time algorithms (simulators) S_1, S_2 such that*

$$\{S_1(1^n, x, f_1(x, y))\} \overset{c}{\equiv} \{view_1^{\pi}(x, y, n)\}$$

$$\{S_2(1^n, y, f_2(x, y))\} \overset{c}{\equiv} \{view_2^{\pi}(x, y, n)\}$$

and correctness is observed, that is for all x, y and n there exists a negligible function μ such that

$$Pr[output^{\pi}(x, y, n) \neq f(x, y)] \leq \mu(n).$$

This concludes the definitions of semi-honest security. In the next section we define security in the malicious model.

To give a flavour of how these definitions can be realised we next consider a simple OT_2^1 protocol and prove it secure in the semi-honest model.

Example 4. *In this example we introduce a simple OT_2^1 protocol and show how it provides security in the semi-honest model. The protocol we present uses a trusted initialiser[6] to distribute co-related randomness to the parties. While this is not always desired, a large area of research in cryptography uses this so-called* trusted initialiser model. *Using a trusted initialiser we are able to prove perfect security for both parties in the protocol, something that is not possible when not using a trusted initialiser. In the protocol below the Sender (P_1) holds two bits (m_0, m_1) and the Receiver (P_2) holds a choice bit σ.*

$$r_0, r_1 \xleftarrow{\$} \{0,1\}, d \xleftarrow{\$} \{0,1\}$$

$$\begin{array}{ccc}
\textit{Sender} & & \textit{Receiver} \\
m_0, m_1 \in \{0,1\}, r_0, r_1 & & \sigma \in \{0,1\}, d, r_d \\
e & \xleftarrow{\qquad e \qquad} & e = \sigma \oplus d \\
f_0 = m_0 \oplus r_e & & \\
f_1 = m_1 \oplus r_{1-e} & \xrightarrow{\qquad f_0, f_1 \qquad} & f_0, f_1 \\
 & & m_\sigma = f_\sigma \oplus r_d
\end{array}$$

In the protocol the trusted initialiser uniformly samples r_0, r_1 and sends them to the Sender, and samples d uniformly and sends it to the Receiver along with r_d. It is r_d that provides the co-related randomness that is important to the protocol. The Receiver masks their choice bit σ by xoring it with d and sends the result to the Sender who then, in the same way, masks the messages with the randomness provided by the trusted initialiser and sends the resulting messages (f_0, f_1) to the Receiver. The Receiver can then learn its choice of message by computing $f_\sigma \oplus r_d$.

Intuitively security for the Sender comes from the fact the messages are only ever sent to the Receiver under the encryption *of being xored with some randomness (r_0 or r_1), meaning the Receiver cannot learn the message for which they do not have the corresponding co-related randomness for. The security of the Receiver is due to their choice bit only being sent to the Sender after being xored with the the uniformly sampled bit d.*

Theorem 6. *The OT_2^1 protocol given in Example 4 is secure in the semi-honest model. In particular we have perfect security for both parties.*

Proof. The functionality we consider here is deterministic therefore we can use the more relaxed definitions of security given in Definition 10. First we show the protocol is correct. The output of the protocol will always equal the output of the functionality: we see this by considering the cases on σ: if $\sigma = 0$ then $f_\sigma \oplus r_d = m_0 \oplus r_e \oplus r_d = m_0 \oplus r_{0 \oplus d} \oplus r_d = m_0$ and if $\sigma = 1$ then $f_\sigma \oplus r_d = m_1 \oplus r_{1-e} \oplus r_d = m_1 \oplus r_{1-(1 \oplus d)} \oplus r_d = m_1$. Using this we have

$$Pr[output^\pi(x,y) \neq f(x,y)] = 0.$$

[6] We note this is often calledd the pre-processing model, or correlated randomness setup model

The security parameter does not feature in the protocol as and therefore we do not include it as an input to $output^{\pi}$.

Next we consider the security of each party in turn. We first assume the Sender is corrupted and show how we can simulate its view of the protocol.

PARTY 1 - THE SENDER The distribution for the real view for the Sender is as follows:

$$\{(m_0, m_1), r_0, r_1, e\}$$

where $r_0, r_1 \overset{\$}{\leftarrow} \{0,1\}$ and $e = \sigma \oplus d$. We construct the following simulator and show it is equal to the real view — the simulator takes in as input (m_0, m_1) (note there is no output in the protocol for the Sender):

1. $r_0, r_1 \overset{\$}{\leftarrow} \{0,1\}$
2. $e \overset{\$}{\leftarrow} \{0,1\}$
3. output $\{(m_0, m_1), r_0, r_1, e\}$

The only difference between the real and simulated view is the last component, one outputs e and the other $d \oplus \sigma$ where $d, e \overset{\$}{\leftarrow} \{0,1\}$. These two distributions are equal due to the one time pad nature of the samples. This combined with the rest of the view being independent of the randomness used to construct the last component means the real and simulated view are equal.

PARTY 2 - THE RECEIVER The distribution for the real view for the Receiver is as follows:

$$\{\sigma, d, r_d, f_0, f_1\}$$

where $r_0, r_1, d \overset{\$}{\leftarrow} \{0,1\}$ and $f_0 = m_0 \oplus r_e$ and $f_1 = m_1 \oplus r_{1-e}$ where $e = \sigma \oplus d$. We construct the following simulator and show it is equal to the real view — the simulator takes in as input σ and m_σ:

1. $r_0, r_1, d \overset{\$}{\leftarrow} \{0,1\}$
2. $f_0, f_1 \overset{\$}{\leftarrow} \{0,1\}$
3. output $\{\sigma, d, r_d, f_0, f_1\}$

The only difference in the real and simulated view are the last two components f_0, f_1. In the simulator they are uniform samples from $\{0,1\}$ and in the real view they are $f_0 = m_0 \oplus r_e$ and $f_1 = m_1 \oplus r_{1-e}$. The distributions on their own are equal, as we are using a one time pad, and as the randomness used for the one time pad is not used elsewhere in the protocol the overall distributions are also equal. Thus we have shown perfect security for Party 2.

□

Example 4 has shown how a protocol for OT_2^1 can satisfy semi-honest security. The protocol uses a trusted initialiser, which is more simple than the protocols we consider in our formalisation later in the thesis.

2.4.2 *Defining Malicious Security*

In the malicious security model an adversary fully corrupts one of the parties and sends all messages on its behalf. There are however adversarial behaviours we cannot account for even in the malicious setting:

1. A party refusing to take part in the protocol.
2. A party substituting their local input — and executing the protocol with an input other than the one provided to them.
3. A party aborting the protocol early.

It is well known the malicious model has these weaknesses. Of these behaviours the second is most interesting. In the malicious setting it is unclear what constitutes a party's *correct* input to a protocol, a corrupted party may enter the protocol with an input that is not equal to its *true* local input. In particular there is no way to tell what the *correct* local input is compared to the input claimed by the party. For further discussion of these limitations see [38, section 7.2.3].

A protocol is said to be secure if the adversary's behaviour is limited to the three actions given above. We consider the malicious security definitions from Goldreich [38, section 7.2.3] and Lindell [44, section 2.3.1] where an ideal and real world are considered. The intuition behind security in the malicious model is as follows: security holds in the ideal world by its construction and use of a trusted party. The idea to prove security is to show the real world is indistinguishable from the ideal world, and is therefore also secure. The three limitations of the malicious model given above are built into the ideal model, that is they are permitted attacks in the ideal world.

In the two party setting the only setting that it is reasonable to consider is when exactly one party is corrupted; if neither party is corrupted then there is no adversary to learn anything anyway, and if both parties are corrupted nothing can be guaranteed.

The ideal model uses a trusted party that ensures security by definition — we let x be the input of Party 1, y be the input of Party 2 and z be the auxiliary input (a priori information) available to the adversary. The ideal model is defined as follows [44]:

Definition (informal) 11. *[Ideal model] An ideal execution of a protocol proceeds as follows.*

- ***Send inputs to trusted party:*** *The honest party sends its received input to the trusted party. The input for the corrupted party is outputted by the adversary and given to the trusted party (it could be abort, the adversary chooses the input based on the original input and z).*
- ***Early abort:*** *If the trusted party receives abort from the corrupted party it sends abort to both parties and the execution is terminated.*
- ***Trusted party computation:*** *The trusted party computes the functionality using the inputs provided by both parties and sends the corrupted party its output.*
- ***Adversary aborts or continues:*** *The adversary, upon receiving its output, instructs the trusted party to abort or continue. If abort is sent the execution is terminated, if continue is sent the trusted party sends the honest party its output.*

- ***Outputs:*** *The honest party outputs the output it received from the trusted party, the corrupted party outputs nothing. The adversary outputs any arbitrary function of the initial input, auxiliary input, and the output given to it by the trusted party.*

Then the ideal execution of the functionality f for party $i \in \{0,1\}$ on inputs (x,y) and auxiliary input z to adversary $\mathcal{A}$ and security parameter n is denoted by $IDEAL_{f,S(z),i}(x,y)$ and defined to be the output pair of the honest party and the adversary in the ideal execution described above.

We next define the execution in the real model.

Definition (informal) 12 (Real model)**.** *The real model for a two party protocol π for party $i \in \{0,1\}$ on inputs (x,y) and auxiliary input z to adversary $\mathcal{A}$ and security parameter n is denoted by is denoted as $REAL_{\pi,\mathcal{A}(z),i}(x,y)$. This is defined as the output pair of the honest party and the adversary from a real execution of π were all messages sent by the corrupted party are constructed by the adversary.*

Having defined the real and ideal worlds we can make the definition of security. Like in the semi-honest setting we require the two worlds to be indistinguishable. Intuitively this means the ideal model can simulate executions of the real model of the protocol.

Definition (informal) 13. *Let π be a two party protocol for computing the functionality f. π is said to securely compute f with abort in the presence of static malicious adversaries if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary S for the ideal model, such that for every $i \in \{1,2\}$ we have,*

$$\{IDEAL_{f,S(z),i}(x,y)\} \overset{c}{\equiv} \{REAL_{\pi,\mathcal{A}(z),i}(x,y)\}.$$

As our formalisation shows, proving security in the malicious model is far more complex than in the semi-honest model. The reason for this is two-fold: firstly, the definition of security is more involved, we must consider a more intricate ideal model. Secondly, protocols that provide malicious security are generally more complex which means reasoning about them is too.

3 INTRODUCTION TO ISABELLE AND CRYPTHOL

In this chapter we introduce Isabelle/HOL and CryptHOL highlighting the parts important to our work. For more detail on Isabelle we point the reader to the book 'Concrete Semantics' by Nipkow and Klein [58] and to the tutorial by Nipkow, Paulson and Wenzel [59]. For more detail on CryptHOL we point the reader to the original paper [50] and the extended version [8], a tutorial [52] and the formalisation [51].

3.1 ISABELLE/HOL

Isabelle/HOL is an interactive theorem prover that implements Higher Order Logic (HOL). HOL is built on simple set-theory, where types are interpreted as sets of elements and terms are elements of the set corresponding to their type. Although Isabelle has a good degree of automation a proof must still be broken down into small steps, from which the user can guide the automation tools to find a proof. Throughout this thesis we do not show our formal proofs, choosing to outline the proof structure instead. The formalisation mostly uses the structured and more human-readable Isar proof language and thus we point the interested reader towards the source theory files in the Archive of Formal Proofs (AFP) [20, 24] for more details.

While all the formal definitions, lemmas and theorems have been mechanically checked by Isabelle we have typeset all formal statements in this thesis manually — thus typing errors may have slipped in.

3.1.1 *Isabelle Notation*

The notations we use in this thesis resemble closely the syntax of Isabelle/HOL (Isabelle). For function application we write $f(x, y)$ in an uncurried form for ease of reading instead of $f\ x\ y$ as in the sources. Basic types include booleans, naturals and integers. To indicate that term t has type τ we write $t\ ::\ \tau$.

Isabelle uses the symbol $\Rightarrow$ for the function type, so $a \Rightarrow b$ is the type of functions that takes an input of type a and outputs an element of type b. The type variable $'a$ denotes an abstract type. The implication arrow $\longrightarrow$ is used to separate assumptions from conclusions inside a HOL statement as is $\Longrightarrow$ however the latter cannot be used inside other HOL formulae. For example modus ponens is written $P \longrightarrow Q \Longrightarrow P \Longrightarrow Q$ which is equivalent to:

$$\frac{P \longrightarrow Q \quad Q}{Q}$$

In HOL a function may be nameless, that is, $\lambda x.\ s(x)$, is the function that maps every value w to the results of s where x is replaced by w. In the situation where s does not depend on x, the underscore _, replaces x in our notation.

Sets, of type *'a set* are isomorphic to predicates of type *'a* $\Rightarrow$ *bool* using the membership map $\in$.

Pairs have the type $'a \times 'b$, the projections of the first and second elements are written *fst* $::$ $'a \times 'b \Rightarrow 'a$ and *snd* $::$ $'a \times 'b \Rightarrow 'b$ respectively. Tuples are nested pairs where nesting associates to the right, for example $(a, b, c) = (a, (b, c))$. The type $'a + 'b$ denotes the disjoint sum, or union type, in this case the injections are *Inl* $::$ $'a \Rightarrow 'a + 'b$ and *Inr* $::$ $'b \Rightarrow 'a + 'b$.

The option datatype

datatype *'a option* = *None* | *Some 'a*

adds the element *None* to the datatype *'a*. All original elements of *'a* are still contained in *'a option*.

One technical aspect of Isabelle we use heavily is the module system, called locales. At a high level locales allow the user to prove theorems abstractly, relative to given assumptions. These theorems can be reused in situations where the assumptions themselves are theorems. The locale declares fixed parameters and assumptions regarding these parameters. The locale name is the predicate that collects all the assumptions made in the locale.

For example the locale for cyclic groups in CryptHOL[1], *cyclic-group*, builds on the locale *group*, whose parameter is the group G and makes two assumptions. The first that the generator is in the carrier set of G, the second that the carrier set of G is contained in the range of the map exponentiation the generator.

$$\begin{aligned}
&\textbf{locale}\ \textit{cyclic-group} = \textit{group}\ G\\
&\quad \textbf{for}\ G :: \ ('a, b)\ \textit{cyclic-group-scheme}\ (\textbf{structure}) +\\
&\quad \textbf{assumes}\ \textit{generator}\ G \in \textit{carrier}(G)\\
&\quad\quad \textbf{and}\ \textit{carrier}(G) \subseteq \textit{range}(\lambda n.\ (\textit{generator}\ G)^n)
\end{aligned} \tag{3.1}$$

where the cyclic group is of record type

$$\begin{aligned}
&\textbf{record}\ '\textit{a cyclic-group} = '\textit{a monoid} +\\
&\quad \textit{generator}\ ::\ 'a
\end{aligned} \tag{3.2}$$

A record of *n fields* is an *n* tuple; they can be updated and extended. The record for cyclic groups extends the record for monoids by adding a generator.

3.2 CRYPTHOL

CryptHOL [8] is a framework for reasoning about *reduction-based* security arguments that is embedded inside the Isabelle/HOL theorem prover. Originally CryptHOL was designed to construct game-based proofs; in this case games are expressed as probabilistic programs that are shallowly embedded in Isabelle.

[1] We heavily use the construction of cyclic groups from CryptHOL as many protocols depend on them.

CryptHOL, like much of modern cryptography, is based on probability theory. Probabilistic programs in CryptHOL are shallowly embedded as subprobability mass functions of type *spmf* using Isabelle's library for discrete distributions. These can be thought of as probability mass functions with the exception that they do not have to sum to one — we can lose some probability mass. This allows us to model failure events and assertions. When a sub probability mass function does sum to one, we say it is lossless.

HOL functions cannot in themselves provide effects like probabilistic choice therefore all such effects are modeled using monads. A monad consists of a (polymorphic) type constructor, in this case *spmf* and two (polymorphic) operations, $return\ ::\ \alpha \Rightarrow \alpha\ spmf$ and $bind\ ::\ \alpha\ spmf \Rightarrow (\alpha \Rightarrow \beta\ spmf) \Rightarrow \beta\ spmf$.

We now introduce the parts of CryptHOL that are relevant for this thesis.

WRITING PROBABILISTIC PROGRAMS Probabilistic programs can be encoded as sequences of functions that compute over values drawn from spmfs. CryptHOL provides some easy-to-read do notation, like in Haskell, to write probabilistic programs, where $\mathbf{do}\{x \leftarrow p;\ f(x)\}$ is the probabilistic program that samples x from the distribution p and returns the *spmf* produced by f when given x. We also sometimes write this as $p \rhd (\lambda\ x.\ f(x))$ as it is less cumbersome than the **do** notation. We can also return an *spmf* using the monad operation *return*. The following probabilistic program, *completeness-game*, is used in our formalisation of the completeness property of Σ-protocols, given in section 4.2. Here *init* and *response* are the probabilistic programs that define the two steps of a Σ-protocol completed by the Prover and *check* is the function that the verifier uses to validate the response. To define the *completeness-game*, *init* and *response* are sampled like in a real execution of a Σ-protocol, and the distribution (*spmf*) of *check* is returned. Note, *check* is deterministic therefore we must return the output as a probability distribution.

$$\begin{aligned} &completeness\text{-}game(h, w, e) = do\ \{\\ &\quad (r, a) \leftarrow init(h, w);\\ &\quad z \leftarrow response(r, w, e);\\ &\quad return(check(h, a, e, z))\} \end{aligned} \tag{3.3}$$

We note that *bind* is commutative, that is, assuming no dependency conditions one can bind spmfs in any order. In particular, given a sequence of samplings the ordering of such samplings is irrelevant.

Under *bind* we also have that constant elements cancel. In particular if p is lossless (its probability mass sums to one), then

$$bind(p, \lambda\_.\ q) = q. \tag{3.4}$$

Our proofs of security are mainly completed by manipulating the appropriate probabilistic programs. While the proofs that each manipulation is valid are not always accessible to non-experts, the effect of each manipulation can be easily seen and recognised as they are explicitly written in the do notation.

ASSERTIONS Making assertions inside probabilistic programs is sometimes useful. For example we must ensure that the adversary in the hiding game (Equation 5.3) outputs two valid messages for the game to proceed. The monad for subprobabilities has an element, $\bot$, that accounts for failure meaning the current part of the probabilistic program is aborted. This is captured by assertion statements

$$assert(b) = if\ b\ then\ return(\_)\ else\ \bot \tag{3.5}$$

where if b holds then the probabilistic program continues otherwise it fails. Here $(\_)$ is the only element of the unit type, returning this element continues with execution of the program with no effect. Assertions are often used in conjunction with the *TRY p ELSE q* construct. For example *TRY p ELSE q* would distribute the probability mass not assigned by p to the distribution according to q. Picking up on our example of the hiding game; if the adversary fails to output two valid messages, the assertion fails and the *ELSE* branch is invoked — resulting in the adversary's output being a coin flip meaning they do not win the resulting security game.

Assertions are not a necessity to our formalisation as the assumptions could be made explicitly in the theorem statements, for example in any statement of the hiding property we could assume all messages outputted by the adversary ($\mathcal{A}_1$) are valid as follows:

$$\forall vk.\ (m_0, m_1) \in set\text{-}spmf(\mathcal{A}_1) \longrightarrow valid\text{-}msg(m_0) \wedge valid\text{-}msg(m_1). \tag{3.6}$$

Here *set-spmf* is the support set of spmf it takes as input. Assertions, in general, make the formalisation more neat and readable.

SAMPLING Sampling from sets is important in cryptography. CryptHOL provides an operation *uniform* which returns a uniform distribution over a finite set. We use two cases of this function extensively: by *samp-uniform*(q), where q is a natural, we denote the uniform sampling from the set $\{0, \dots, q-1\}$ and by *coin* we denote the uniform sampling from the set $\{True, False\}$ — a coin flip.

The monad operations give rise to another function, $map :: (\alpha \Rightarrow \beta) \Rightarrow \alpha\ spmf \Rightarrow \beta\ spmf$.

$$map(f, p) = bind(p, (\lambda x.\ return(f(x)))) \tag{3.7}$$

The map function can be thought of as the *post-processing* of sampled values. It is from this level of abstraction that we are able to reason about the equivalence of distributions and thus complete major steps in our proofs. For example, we can apply one time pad lemmas. Below is that statement of the one time pad for addition in the finite group $\mathbb{Z}_q$.

$$map((\lambda b.\ (y + b)\ mod\ q), (samp\text{-}uniform(q))) = samp\text{-}uniform(q) \tag{3.8}$$

PROBABILITIES Security definitions are based on explicit probabilities of events occurring. In CryptHOL the expression $\mathcal{P}[Q = x]$ denotes the subprobability mass the spmf Q assigns to the event x. In our proofs reasoning at this level is often the last step, much of the proof effort is in showing properties of the probabilistic programs over which the probabilities are defined.

NEGLIGIBLE FUNCTIONS To reason about security in the asymptotic case we must consider negligible functions. These are formalised as a part of CryptHOL in the canonical way. A function, $f\ ::\ nat \Rightarrow real$, is said to be negligible if

$$\forall c > 0.\ f \in o(\lambda x.\ inverse(x^c)) \tag{3.9}$$

where o is the little o notation. We discuss the use of such functions in our proofs in sections 6.3.2 and 7.3.1.4 where we consider the asymptotic security setting.

CYCLIC GROUPS We highlight the formalisation of cyclic groups that CryptHOL provides; the construction provides the user with a cyclic group G and a generator g. The formalisation extends the formalisation of monoids in Isabelle/HOL meaning there is an armoury of lemmas immediately available for use. We use cyclic groups in the formalisation of the Pedersen commitment scheme and the Schnorr, Chaum-Pedersen and Okamoto Σ-protocols. In the formal parts of this thesis we denote group multiplication by $\otimes$ whereas we denote the multiplication of natural numbers by $\cdot$. In the informal parts all multiplication is written as $\cdot$.

3.3 FORMALISATION OVERVIEW

CryptHOL has been used for a number of formalisations of cryptography thus far. Our work lends weight to the fact that CryptHOL provides a good environment for such formalisations, in particular that the method of modularisation can be used for considering low level cryptographic primitives.

In this section we first discuss the general method of our formalisation at a high level, in particular how CryptHOL allows the user to make their definitions abstract and then instantiate them for the proofs we consider. To illustrate this we look at a running example for Σ-protocols; we only introduce this example briefly to get across the key concepts. As a result of the ability to reason abstractly we feel the method we use could be considered as the most general method that Isabelle and CryptHOL allow for. Second we discuss asymptotic security statements in CryptHOL.

3.3.1 *Method of formalisation*

Isabelle's module system and CryptHOL's monadic structure allow for a natural hierarchy in our formalisation. We begin our formalisations by abstractly defining the security properties required. This part of the formalisation is defined over abstract types, giving the flexibility for it to be instantiated for any protocol. The *human reader* needs to only check the high level, abstract, definitions of security to have confidence in the whole collection of proof as all instantiated proofs are made with respect to these definitions. We are also able to prove some lemmas at the abstract level and have them at our disposal in any instantiation, thus reducing the workload for future proofs. Some of the properties are technical and uninteresting to the cryptographer, for example we prove losslessness of various probabilistic programs used in the definitions, however we are also able to reason about more interesting properties. For example, to formalise the construction of commitment schemes from Σ-protocols we work at an abstract level (chapter 6), only assuming the existence of a Σ-protocol. This means the instantiated proofs (for the concrete Σ-protocols we consider) come for free once we prove they are Σ-protocols.

We next more explicitly describe the workflow in constructing our formalisation. Here we consider Σ-protocols as a running example. First we show the initial abstract locale, then how we use this to define completeness, and finally how we instantiate the abstract locale for the Schnorr Σ-protocol.

3.3.1.1 *Instantiating the abstract frameworks*

We use Isabelle's locales to define properties of security relative to fixed constants and then instantiate these definitions for explicit protocols and prove the security properties as theorems.

1. To consider Σ-protocols abstractly and define completeness we fix in a locale the probabilistic programs (algorithms) that make up the primitive (i.e. *init*, *response*, *check*) as well as other parameters of the Σ-protocol. For example the relation *Rel* and S_{raw}, the simulator for the HVZK property. We also make any assumptions we require on the parameters.

$$
\begin{aligned}
&\textbf{locale } \mathit{\Sigma\text{-}protocol\text{-}base} = \\
&\quad \textbf{fixes } \mathit{init} :: (\mathit{'pub\text{-}input} \times \mathit{'witness}) \Rightarrow (\mathit{'rand} \times \mathit{'msg})\ \mathit{spmf} \\
&\qquad \textbf{and } \mathit{response} :: \mathit{'rand} \Rightarrow \mathit{'witness} \Rightarrow \mathit{'challenge} \Rightarrow \mathit{response\ spmf} \\
&\qquad \textbf{and } \mathit{check} :: \mathit{'pub\text{-}input} \Rightarrow \mathit{'msg} \Rightarrow \mathit{'challenge} \Rightarrow \mathit{'response} \Rightarrow \mathit{bool} \\
&\qquad \textbf{and } \mathit{Rel} :: (\mathit{'pub\text{-}input} \times \mathit{'witness})\ \mathit{set} \\
&\qquad \textbf{and } S_{raw} :: \mathit{'pub\text{-}input} \Rightarrow \mathit{'challenge} \Rightarrow (\mathit{'msg}, \mathit{'response})\ \mathit{sim\text{-}out\ spmf} \\
&\qquad \textbf{and } \mathcal{A}_{ss} :: (\mathit{'pub\text{-}input}, \mathit{'msg}, \mathit{'challenge}, \mathit{'response}, \mathit{'witness})\ \mathit{prover\text{-}adversary} \\
&\qquad \textbf{and } \mathit{challenge\text{-}space} :: \mathit{'challenge\ set} \\
&\qquad \textbf{and } \mathit{valid\text{-}pub} :: \mathit{'pub\text{-}input\ set} \\
&\quad \textbf{assumes } \mathit{Domain}(\mathit{Rel}) \subseteq \mathit{valid\text{-}pub}
\end{aligned}
\tag{3.10}
$$

2. Using these fixed parameters we make the required definitions. Below is the definition we construct for completeness. We construct the the probabilistic program *completeness-game*, given previously in Equation 3.3 and use it to define the completeness property.

$$
\begin{aligned}
\mathit{completeness} = (\forall h\ w\ e.\ (h, w) \in \mathit{Rel} \longrightarrow e \in \mathit{challenge\text{-}space} \\
\longrightarrow \mathcal{P}[\mathit{completeness\text{-}game}(h, w, e) = \mathit{True}] = 1)
\end{aligned}
$$

Here we say the Σ-protocol is complete if for all valid challenges the completeness game returns true.

3. To instantiate a Σ-protocol and prove it is complete we explicitly define the fixed parameters from the locale, *Σ-protocol-base*. To do this we refine the types and define the probabilistic programs that describe the protocol.[2] In the case of the Schnorr Σ-protocol we work with a cyclic group G by fixing it in the locale *schnorr-base*.

$$\begin{aligned} &\textbf{locale}\ \textit{schnorr-base} = \\ &\quad \textbf{fixes}\ G\ ::\ \textit{'grp cyclic-group} \\ &\quad \textbf{assumes}\ \textit{prime}(|G|) \end{aligned} \tag{3.11}$$

Inside this locale we define the instantiated parameters: $\textit{init}^S, \textit{response}^S, \textit{check}^S, \textit{Rel}^S, S_{raw}, \mathcal{A}_{ss}, \textit{challenge-space}^S$ and $\textit{valid-pub}^S$ — here the superscript S denotes they are the parameters for the Schnorr protocol.

For example $\textit{init}^S$ is defined as follows:

$$\begin{aligned} &\textit{init}^S\ ::\ (\textit{'grp pub-in} \times \textit{witness}) \Rightarrow (\textit{rand} \times \textit{'grp msg})\ \textit{spmf} \\ &\textit{init}^S(h, w) = \textit{do}\ \{ \\ &\quad r \leftarrow \textit{samp-uniform}(|G|); \\ &\quad \textit{return}(r, g^r)\} \end{aligned} \tag{3.12}$$

4. We then utilise Isabelle's locale structure by importing the abstract theory using the **sublocale** command. This establishes the current context as an interpretation of the abstract theory.

$$\begin{aligned} &\textbf{sublocale}\ \textit{schnorr-}\Sigma :\ \Sigma\textit{-protocol-base}\ \textit{init}^S\ \textit{response}^S\ \textit{check}^S \\ &\qquad\qquad \textit{Rel}^S\ S_{raw}\ \mathcal{A}_{ss}\ \textit{challenge-space}^S\ \textit{valid-pub}^S \end{aligned}$$

Not only must the explicit definitions be of the correct type when importing a locale, one must also discharge any assumptions that come with the locale. In this case we must prove that $\textit{Domain}(\textit{Rel}^S) \subseteq \textit{valid-pub}^S$. Once this has been done our instantiation is valid with respect to the *Σ-protocol-base* locale and we can use its definition of correctness.

5. Any use of a definition from the original locale (in this case *Σ-protocol-base*) requires the definition name to be prefixed by the name we give to the sublocale (in this case *Schnorr-Σ*). The statement of completeness for the Schnorr Σ-protocol is now given by *schnorr-Σ.completeness*.

[2] We note we say we *refine* the types rather than make them explicit. By refine we mean we make the types more explicit but not to the bitstring level, as would be the case in a real implementation of the protocol. For example we may instantiate the abstract type *'witness* to be a natural or the *'pub-input* to be a group element. Note that the cyclic group instantiation is still in many ways abstract as we parameterise over a fixes cyclic group — see the locale in 3.11.

This has given the full workflow of our formalisation process. Throughout this method we do not consider the security parameter, instead assuming it is implicit in all algorithms. We are able to instantiate the security parameter with ease after the proofs in the concrete setting are complete. We discuss this in the next section.

3.3.2 *Polynomial Runtime*

Many of the definitions given in chapter 2 involve the adversary or simulator being bounded to run in polynomial time. This computational bound ensures that adversaries and simulators considered are (in principle) feasible.

Unfortunately, CryptHOL cannot reason about computational runtimes, due to the shallow embedding. We therefore cannot fully formalise notions like computational binding (Definition 7) that quantify over computationally bounded adversaries. Instead, we capture the underlying reduction argument with a reduction-based security theorem. As an example, for constructing a commitment scheme from a Σ-protocol, the concrete security theorem has the following form: the binding advantage *bind-adv*$(\mathcal{A})$ of an adversary $\mathcal{A}$ is bounded by the advantage of a different adversary $\mathcal{A}'$ against the hardness of the underlying relation *Rel*. This adversary $\mathcal{A}'$ is obtained by a reduction f, which systematically transforms binding-game adversaries $\mathcal{A}$ into hardness game adversaries $\mathcal{A}' = f(\mathcal{A})$.

The example above concerns the game-based setting but an analogous argument can be used in the simulation-based setting. Let us assume a distinguisher can distinguish the real and ideal view for a party, then we show that this distinguisher can be used to construct an adversary that can beat a known hard problem. Thus reducing the security to the hard problem.

Such a reduction-based statement captures the key aspects of the security proof. Compared to a computational statement, which quantifies over all computationally bounded adversaries, the reduction f shows up in the security statement itself. This makes the statement more generic in the sense that we need not commit to a particular computational model or complexity class such as polynomial time. Conversely, the reader must manually check that the reduction lies in the desired complexity class.

3.3.3 *Concrete vs. asymptotic security*

In our formalisations, we first prove *concrete* security bounds using reduction-style proofs. That is, we bound on adversary's advantage as a function of advantages of different adversaries of the primitives used in the construction. For example, we show in Lemma 22 in section 6.2.1 that the binding advantage for commitment schemes constructed from Σ-protocols is bounded by the advantage that the (transformed) adversary breaks the hard relation *Rel*. This is in line with other CryptHOL formalisations [21, 50].

From these concrete statements, we can easily derive more abstract asymptotic security statements. To that end, a security parameter must be introduced. We describe in sections 7.3.1.4 and 6.3.2 how we achieve this with little effort using Isabelle's locale system. Conceptually, this process replaces a locale parameter such as the cyclic group $\mathcal{G}$ :: *'grp cyclic-group* with a family of cyclic groups $\mathcal{G}$ :: *nat* ⇒ *'grp cyclic-group*. And similarly, the challenge space *challenge-space* becomes a family of type *nat* ⇒ *'challenge set*. This parameterisation is also the reason for the locale parameters *valid-pub* and *challenge-space*. Since HOL does not have dependent types, the same abstract type *'challenge* must hold the challenge spaces for every possible security parameter value. The parameter *challenge-space* then carves out the right challenge space for the chosen security parameter.

Considering the general example given in section 3.3.2 one can see that such statements naturally yield asymptotic security statements of the following form: The binding advantage of a family of adversaries $\mathcal{A}_\eta$ against the commitment scheme is negligible if the family of reduced adversaries $f(\mathcal{A}_\eta)$ has negligible advantage against the hardness of the underlying relation.

We show how the security parameter is instantiated in two places in this thesis. First in section 7.3.1.4 we show how we instantiate it for the OT^1_2 protocol constructed from ETPs and second in section 6.3.2 we show how it is done for the Pedersen commitment scheme. While the process of formalisation in both cases is analogous we feel by providing two examples, in different contexts, we allow the reader to understand more fully how it is done. In our formalisations we provide proofs in the asymptotic setting for all the protocols we consider.

We do not believe the instantiation of the security parameter in this work is an overly technical contribution, in fact we follow Lochbihler's technique for achieving it. However we do feel that it is an important step to take. It allows the formalised security statements to be closer to the security statements proven in the literature. One goal of formalising results such as these is to as accurately as possible capture them in the formal language, by taking the step to consider asymptotically we have taken one step closer to achieving this.

Part I

FORMALISING Σ-PROTOCOLS AND COMMITMENTS

4 Σ-PROTOCOLS

4.1 INTRODUCTION

In this chapter we first show how we formally define Σ-protocols and their security properties and then how we instantiate our definitions for the protocols and constructions we consider.

CHAPTER OUTLINE First, In Section 4.2 we show our formalisation of the definitions of security for Σ-protocols. Then in Section 4.3 we show how we instantiate various Σ-protocols and constructions with respect our definitions. In Section 4.3.1 we formalise compound Σ-protocols. Then we show how we formalise the Schnorr, Chaum-Pedersen and Okamoto Σ-protocols in Sections 4.3.2, 4.3.3 and 4.3.4 respectively. The workflow for this chapter is depicted in Figure 4.1.

The work in this chapter has been published in [22, 24, 25].

4.2 FORMALISING THE DEFINITIONS

In this section we detail our formalisation of Σ-protocols based on the definitions from section 2.2.

As explained in Section 3.3.1, our method of formalisation is to define a locale where we fix the parameters required for the definitions of Σ-protocols. We do this in the locale *Σ-protocol-base*, given in Statement 4.4. The locale fixes the components of the Σ-protocol. These are:

- *init*: constructs the initial message sent from the Prover to the Verifier, and its corresponding randomness.
- *response*: the response sent from the Prover to the Verifier.
- *check*: performs the verification the Verifier runs on the response from the Prover.

We also fix the relation *Rel* that the protocol is based on, the adversary $\mathcal{A}_{ss}$ required in the special soundness definition, the *challenge-space* which is the set of all possible

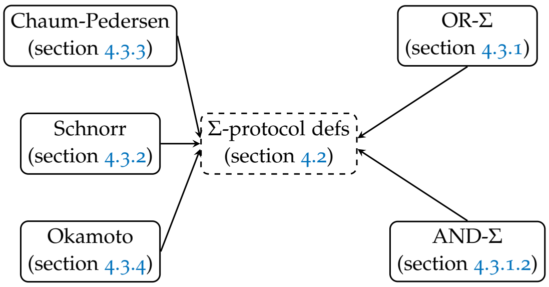


Figure 4.1: Outline of our formalisation of Σ-protocols.

challenges and the set *valid-pub* which contains all the valid public inputs. Finally we require a simulator for the HVZK definition. We take time to discuss our modelling of the simulator here. The simulator outputs a conversation of the form (a,e,z), however a subtlety is that the output challenge e must be the same as the input challenge e; overall the simulator looks as follows:

$$(a,e,z) \leftarrow S(h,e).$$

To formally model this we fix in the locale the part of the simulator, S_{raw}, that constructs a and z and then define the full simulator that outputs (a,e,z) using S_{raw} as follows:

$$S(h,e) = map(\lambda\ (a,z).\ (a,e,z), S_{raw}(h,e)).$$

We do this to ensure the challenge outputted by the simulator is the same as the challenge it takes as input.

To improve the readability of the formalisation we define three type synonyms; the first defines the type of a conversation, the second the output type of S_{raw} and the third the type of the special soundness adversary. Type synonyms have no semantic meaning, our reason for constructing them it mainly to make the locale given in 4.4 more readable as we can give names to the synonyms.

$$\textbf{type-synonym}\ (\text{'}msg, \text{'}challenge, \text{'}response)\ conv\text{-}tuple = \\ (\text{'}msg \ \times\ \text{'}challenge \ \times\ \text{'}response) \tag{4.1}$$

$$\textbf{type-synonym}\ (\text{'}msg, \text{'}response)\ sim\text{-}out = (\text{'}msg \ \times\ \text{'}response) \tag{4.2}$$

$$\textbf{type-synonym} \\ (\text{'}pub\text{-}input, \text{'}msg, \text{'}challenge, \text{'}response, \text{'}witness)\ prover\text{-}adversary \\ = \text{'}pub\text{-}input \Rightarrow (\text{'}msg, \text{'}challenge, \text{'}response)\ conv\text{-}tuple \\ \Rightarrow (\text{'}msg, \text{'}challenge, \text{'}response)\ conv\text{-}tuple \Rightarrow \text{'}witness\ spmf \tag{4.3}$$

The locale for these fixed parameters, *Σ-protocol-base*, is given in 4.4 — note this is the same as the locale given in the running example in section 3.3.

$$\begin{aligned}
&\textbf{locale}\ \Sigma\text{-}protocol\text{-}base = \\
&\quad \textbf{fixes}\ init :: (\text{'}pub\text{-}input \times \text{'}witness) \Rightarrow (\text{'}rand \times \text{'}msg)\ spmf \\
&\qquad \textbf{and}\ response\ ::\ \text{'}rand \Rightarrow \text{'}witness \Rightarrow \text{'}challenge \Rightarrow \text{'}response\ spmf \\
&\qquad \textbf{and}\ check\ ::\ \text{'}pub\text{-}input \Rightarrow \text{'}msg \Rightarrow \text{'}challenge \Rightarrow \text{'}response \Rightarrow bool \\
&\qquad \textbf{and}\ Rel\ ::\ (\text{'}pub\text{-}input \times \text{'}witness)\ set \\
&\qquad \textbf{and}\ S_{raw}\ :: \text{'}pub\text{-}input \Rightarrow \text{'}challenge \Rightarrow (\text{'}msg, \text{'}response)\ sim\text{-}out\ spmf \\
&\qquad \textbf{and}\ \mathcal{A}_{ss}\ ::\ (\text{'}pub\text{-}input, \text{'}msg, \text{'}challenge, \text{'}response, \text{'}witness)\ Prover\text{-}adversary \\
&\qquad \textbf{and}\ challenge\text{-}space\ ::\ \text{'}challenge\ set \\
&\qquad \textbf{and}\ valid\text{-}pub\ ::\ \text{'}pub\text{-}input\ set \\
&\quad \textbf{assumes}\ Domain(Rel) \subseteq valid\text{-}pub
\end{aligned} \tag{4.4}$$

The assumption requires that the domain of the relation is contained in the set of valid public inputs, put another way all elements that are in the relation are valid public inputs.

Next, the language L_R for the relation R is the set of all public inputs for which a witness exists such that the relation holds.

$$L_R = \{x.\ \exists w.\ Rel(x, w)\} \tag{4.5}$$

Using the parameters we fixed in the locale *Σ-protocol-base* we define the properties of Σ-protocols. First we define completeness. For this property the probabilistic program, *completeness-game*, runs the components of the protocol and outputs the output of *check*. We repeat the definition from Equation 3.3.

$$\begin{aligned}
&completeness\text{-}game(h, w, e) = do\ \{\\
&\quad (r, a) \leftarrow init(h, w);\\
&\quad z \leftarrow response(r, w, e);\\
&\quad return(check(h, a, e, z))\}
\end{aligned} \tag{4.6}$$

The definition of completeness is quantified over all public inputs, witnesses and challenges.

Definition 1 (Completeness)**.**

$$\begin{aligned}
completeness = (\forall h\ w\ e.\ (h, w) \in Rel \longrightarrow e \in challenge\text{-}space\\
\longrightarrow \mathcal{P}[completeness\text{-}game(h, w, e) = True] = 1)
\end{aligned}$$

The definition of HVZK follows the simulation-based paradigm: we require the output distribution of the simulator S to be equal to the output distribution of the real view of the protocol which is given below.

$$\begin{aligned}
&real\text{-}view(h, w, e) = do\ \{\\
&\quad (r, a) \leftarrow init;\\
&\quad z \leftarrow response(r, w, e);\\
&\quad return(a, c, z)\}
\end{aligned} \tag{4.7}$$

The real view can be defined abstractly as we know the structure of the protocol. This is unlike in general MPC protocols [21] where the real view has to be defined for each MPC protocol considered. We must still construct a simulator for each instantiated Σ-protocol. As described in Definition 3, we additionally require that the simulator's output produces an accepting conversation even if the public input h does not belong to the language.

Definition 2 (Honest Verifier Zero Knowledge)**.**

$$\begin{aligned}
&HVZK = (\forall e \in challenge\text{-}space.\\
&\quad (\forall (h, w) \in Rel.\ real\text{-}view(h, w, e) = S(h, e))\\
&\qquad \wedge (\forall h \in valid\text{-}pub.\ \forall (a, z) \in set\text{-}spmf(S_{raw}(h, e)).\ check(h, a, e, z)))
\end{aligned}$$

For special soundness to hold we require the special soundness adversary ($\mathcal{A}_{ss}$) to output the witness when given two accepting conversations (with distinct challenges) with respect the public input h, (a, e, z) and (a, e', z'). An accepting conversation is a tuple upon which *check* is satisfied. To capture this formally we must show that for all w' in the support set (*set-spmf*) of $\mathcal{A}_{ss}$ the relation is satisfied. Together with this we require that $\mathcal{A}_{ss}$ is lossless, if not $\mathcal{A}_{ss}$ may output nothing leaving no way to reason about all outputs of $\mathcal{A}_{ss}$.

Definition 3 (Special Soundness)**.**

$$\begin{aligned}
&\textit{special-soundness} = (\forall h\ a\ e\ z\ e'\ z'.\ h \in \textit{valid-pub} \\
&\quad \longrightarrow e \in \textit{challenge-space} \longrightarrow e' \in \textit{challenge-space} \longrightarrow e \neq e' \\
&\qquad \longrightarrow \textit{check}(h, a, e, z) \longrightarrow \textit{check}(h, a, e', z') \longrightarrow \\
&\qquad\quad \textit{lossless}(\mathcal{A}_{ss}(h, (a, e, z), (a, e', z'))) \wedge \\
&\qquad\qquad \forall w' \in \textit{set-spmf}(\mathcal{A}_{ss}(h, (a, e, z), (a, e', z'))).\ \textit{Rel}(h, w'))
\end{aligned}$$

Using these three definitions we define the notion of a Σ-protocol.

Definition 4 (Σ-protocol)**.**

$$\Sigma\textit{-protocol} = \textit{completeness} \wedge \textit{special-soundness} \wedge \textit{HVZK}$$

It may appear surprising that in our formalisation of Σ-protocols we do not fix a probabilistic program to output the challenge, like we do for the other components of the protocol. In this case it is not needed as the Verifier, who outputs the challenge, is assumed to be honest. In particular we define the properties over all allowed challenges ($\forall e \in \textit{challenge-space}$). This is valid when the challenge is always generated honestly, however is not strong enough if the challenge was not generated honestly — in the case of a corrupt Verifier. This extension is considered by full Zero-Knowledge protocols [16], which we do not consider in this work.

4.2.1 *Differences in the definitions of Σ-protocols*

There are different definitions of Σ-protocols presented in the literature [7, 28, 29, 31, 44]. We now discuss this and the consequences of Cramer's additional HVZK requirement (Condition 2 in Definition 3). In particular we show how we highlight the correct definition of Σ-protocols from the literature. We also outline how Barthe et al. dealt with this issue in their formalisation of Σ-protocols [7].

DAMGÅRD'S HVZK DEFINITION Damgård's definition [31] of HVZK does not require the inputs to the real view to satisfy the relation, it only requires that the output distributions of the simulator and real view are equal. We found two problems with this requirement. First, the real view is not well-defined if the public input is not in the relation: to construct the real view, we must run the Prover and the Prover runs only if it gets a witness as input, but there is no such witness when the public input is not in the relation. Accordingly, none of the proofs of HVZK for Σ-protocols we study work. For example, without the assumption that $h = g^w$ (from $(h, w) \in Rel^S$) in the

Schnorr Σ-protocol, we cannot reason about the real view and the simulator being equal. Second, Damgård assumes in the proofs in [31] that the relation holds for the input. We therefore conclude that Damgård probably intended that $(h, w) \in \mathit{Rel}$ in his definition.

HAZAY'S AND LINDELL'S HVZK DEFINITION In [44], Hazay and Lindell credit Damgård for providing the 'basis' of their presentation of Σ-protocols. Their definition requires the relation to be satisfied on the public input and witness that are inputs to the real view. This corresponds to Condition 1 of Definition 3 in this work.

Damgård [31] and Hazay and Lindell [44] both carry out the OR construction for Σ-protocols with the relation Rel_{OR} as defined in section 4.3.1.1, with a proof similar to ours. However, their proofs are flawed as the simulator for the HVZK property is unspecified for public inputs h that are not in the language. Accordingly, completeness need not hold. For every Σ-protocol we have encountered this property holds. It must be included in the definition as when dealing with an arbitrary Σ-protocols in constructions like the OR construction the property is needed for the proofs.

CRAMER'S HVZK DEFINITION Cramer [29] additionally requires that the simulator outputs an accepting conversation when the public input is not in the language, which corresponds to Condition 2 in 3. This ensures that the completeness proof of the OR construction for Σ-protocols goes through. Lindell has confirmed that it was implicitly assumed in his proof [private communication, 2019]. We therefore conclude that the extended definition that we gave should be the standard one.

To our knowledge no real-world Σ-protocol violates the additional requirement — pathological examples can of course be constructed, yet this extended property is rarely mentioned in the literature.

BARTHE ET AL.'S FORMALISATION AND CIAMPI ET AL.'S HVZK DEFINITION There is another way to rescue the OR construction without adding Cramer's requirement, namely changing the definition of Rel_{OR}. Barthe et al. [7] also noticed the completeness issue for the OR construction in their formalisation of Σ-protocols. They recovered the proof by defining Rel_{OR} as

$$\begin{aligned} \mathit{Rel}_{OR} = \{((x_0, x_1), w).\ &((x_0, w) \in \mathit{Rel}_0 \wedge x_1 \in \mathit{Domain}(\mathit{Rel}_1)) \\ &\vee ((x_1, w) \in \mathit{Rel}_1 \wedge x_0 \in \mathit{Domain}(\mathit{Rel}_0))\}, \end{aligned} \tag{4.8}$$

i.e., both the inputs x_0 and x_1 must be in the language. Ciampi et al. [28] use the same definition in their paper proofs.

In contrast, our definition (and Damgard's, Hazay's and Lindell's, and Cramer's) requires only one input x_0 or x_1 to be in the language; the other need only meet syntactic constraints as formalised by *valid-pub*. This small difference has a substantial impact on the expressive power of the OR construction. With (4.8), the languages for the constituent Σ-protocols must be *efficiently* decidable. Indeed, Ciampi et al. "implicitly assume that the Verifier of a protocol for relation R executes the protocol only if the common input x belongs to L_R and rejects immediately common inputs not in L_R" [28]. For relations like the discrete logarithm, this is not a problem because every group element has a discrete logarithm; the hard part is computing it. However, there are Σ-protocols where the language itself is hard, e.g., Blum's protocol for a Hamiltonian cycle in a graph [15]. The OR construction with the relation (4.8) does not work for such Σ-protocols.

4.3 Σ-PROTOCOL INSTANTIATIONS AND CONSTRUCTIONS

In this section we use the framework we defined in section 4.2 to instantiate Σ-protocols and prove they are secure. First, in section 4.3.1, we consider the construction of compound Σ-protocols and then in the remaining sections we consider the well known Σ-protocols of Schnorr, Chaum-Pedersen and Okamoto.

4.3.1 *Compound Σ-protocols*

Σ-protocols can be combined to prove knowledge for AND and OR statements. Consider two Σ-protocols, Σ_0 and Σ_1, with relations Rel_0 and Rel_1 respectively. The AND construction allows the Prover to prove they know witnesses w_0 and w_1 such that both $Rel_0(x_0, w_0)$ and $Rel_1(x_1, w_1)$ are true and the OR construction allows for the proof of knowledge of a witness such that $Rel_0(x_0, w)$ or $Rel_1(x_1, w)$ is true — (x_0, x_1) is the public input. Cryptographers have found many uses for these basic constructions, for example the voting protocols in [29]. In this section we detail our formalisation of both constructions.

4.3.1.1 *The OR construction*

The construction of the OR protocol follows the idea that the Prover can run the real protocol for the relation for which the witness is known and run the simulator to generate the conversation for the relation for which the witness is not known. By the HVZK property of Σ-protocols the simulated view is equivalent to the real view, therefore the Verifier cannot tell which was constructed by the real protocol and which from the simulator. The protocol is shown in Figure 4.2.

In the literature [29, 31, 44] the OR construction is considered over bitstrings. However we only require the one time pad property of the xor function thus we are able to generalise the construction to arbitrary boolean algebras. To do this we formalise the concept of a boolean algebra[1] and prove the one time pad property, whose statement is seen in Equation 4.9.

$$map((\lambda a.\ a \oplus x), (uniform(carrier(L))) = uniform(carrier(L)) \tag{4.9}$$

where L is the boolean algebra with xor function $\oplus$.

To formalise the OR construction we fix two Σ-protocols (Σ_0 and Σ_1) and their respective components

$$init_i,\ response_i,\ check_i,\ Rel_i,\ S_{raw,i},\ \mathcal{A}_{ss,i},\ challenge\text{-}space_i,\ valid\text{-}pub_i$$

[1] The formalisation of a boolean algebra was done by Andreas Lochbihler and is contained in our AFP entry [24]. This part of the formalisation is not extensive and consists of 227 lines.

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier} \\
(x_0, x_1), w & & (x_0, x_1) \\
(r_b, a_b) \overset{\$}{\leftarrow} init_b(x_b, w) & & \\
e_{1-b} \overset{\$}{\leftarrow} L & & \\
(a_{1-b}, e_{1-b}, z_{1-b}) \overset{\$}{\leftarrow} S_b(x_{1-b}, e_{1-b}) & \xrightarrow{\text{initial msg: } (a_0, a_1)} & (a_0, a_1) \\
s & \xleftarrow{\text{challenge: } s} & s \overset{\$}{\leftarrow} L \\
e_b = s \oplus e_{1-b} & & \\
z_b \leftarrow response_b(r_b, w, e_b) & \xrightarrow{\text{response: } (e_0, z_0, e_1, z_1)} & (e_0, z_0, e_1, z_1) \\
 & & \text{checks: } s = e_0 \oplus e_1 \\
 & & check_0(x_0, a_0, e_0, z_0) \\
 & & check_1(x_1, a_1, e_1, z_1)
\end{array}$$

Figure 4.2: The OR construction for two Σ-protocols, Σ_0 and Σ_1. L is the boolean algebra that the protocol is run over. (x_0, x_1) is the public input such that $Rel_0(x_0, w)$ or $Rel_1(x_1, w)$ is satisfied and b represents the relation that holds, that is we have that $Rel_b(x_b, w)$. n this section we denote the challenge as s to distinguish it from the challenges of the underlying Σ-protocols which we will denote with e_0 and e_1.

for $i \in \{0, 1\}$ as well as a boolean algebra L :: *'bool-alg boolean-algebra*. The only type constraint on the components of Σ_0 and Σ_1 is that both challenges must be of type *'bool-alg*. We allow the types of Σ_0 and Σ_1 to be different, thus the witness must be a sum type w :: $('witness_0 + 'witness_1)$.

We define the relation,

$$Rel_{OR} :: (('pub_0 \times 'pub_1) \times ('witness_0 + 'witness_1))\ set \tag{4.10}$$

as an inductive set with the following introduction rules:

$$((x_0, x_1), Inl(w_0)) \in Rel_{OR} \textbf{ if } (x_0, w_0) \in Rel_0 \wedge x_1 \in valid\text{-}pub_1 \tag{4.11}$$

$$((x_0, x_1), Inr(w_1)) \in Rel_{OR} \textbf{ if } (x_1, w_1) \in Rel_1 \wedge x_0 \in valid\text{-}pub_0 \tag{4.12}$$

In particular the Prover knows a witness for one of the two relations, and knows to which relation the witness belongs to. We also require that the public input for which the Prover does not know the witness is a valid public input for its respective Σ-protocol.

In the OR construction the initial message is constructed as either the real initial message (of the Σ-protocol for which the Prover knows the witness) or the first message of the simulator (of the other Σ-protocol). The probabilistic program $init_{OR}$

has an output consisting of two parts: 1. the randomness consisting of the randomness from $init_b$ (where $b \in \{0,1\}$ is the relation for which the Prover knows the witness), the random challenge sampled, as well as the response from the conversation that is simulated and 2. the initial messages sent in the protocol, one (and only one) of which is constructed by the simulator.

$$\begin{aligned}
&init_{OR}((x_0, x_1), Inl(w_0)) = do \, \{ \\
&\quad (r_0, a_0) \leftarrow init_0(x_0, w_0); \\
&\quad e_1 \leftarrow uniform(carrier(L)); \\
&\quad (a_1, e_1, z_1) \leftarrow S_1(x_1, e_1); \\
&\quad return(Inl(r_0, e_1, z_1), (a_0, a_1)\}
\end{aligned} \tag{4.13}$$

$$\begin{aligned}
&init_{OR}((x_0, x_1), Inr(w_1)) = do \, \{ \\
&\quad (r_1, a_1) \leftarrow init_1(x_1, w_1); \\
&\quad e_0 \leftarrow uniform(carrier(L)); \\
&\quad (a_0, e_0, z_0) \leftarrow S_0(x_0, e_0); \\
&\quad return(Inr(r_1, e_0, z_0), (a_0, a_1))\}
\end{aligned} \tag{4.14}$$

The return type of $init_{OR}$ is

$$((((\text{'}rand_0 \times \text{'}bool \times \text{'}response_1 + \text{'}rand_1 \times \text{'}bool \times \text{'}response_0)) \times \text{'}msg_0 \times \text{'}msg_1)) \; spmf$$

where *'bool* is the type of the boolean algebra.

To respond to a challenge, s, the Prover constructs a new challenge to be used in constructing the real response by xoring it with the challenge e it generated in $init_{OR}$. The response for the relation the Prover does not know is given as the simulated response from the $init_{OR}$ phase. The inputs to $response_{OR}$ consist of (1). the randomness outputted by $init_{OR}$ (a 3-tuple) (2). the witness that is known and (3). the challenge.

$$\begin{aligned}
&response_{OR}(Inl(r_0, e_1, z_1), Inl(w_0), s) = do \, \{ \\
&\quad let \; e_0 = s \oplus e_1; \\
&\quad z_0 \leftarrow response_0(r_0, w_0, e_0); \\
&\quad return((e_0, z_0), (e_1, z_1))\}
\end{aligned} \tag{4.15}$$

$$\begin{aligned}
&response_{OR}(Inr(r_1, e_0, z_0), Inr(w_1), s) = do \, \{ \\
&\quad let \; e_1 = s \oplus e_0; \\
&\quad z_0 \leftarrow response_1(r_1, w_1, e_1); \\
&\quad return((e_0, z_0), (e_1, z_1))\}
\end{aligned} \tag{4.16}$$

To check the responses given by the Prover, the Verifier checks both conversations it receives are valid with respect the Σ-protocols they correspond to as well as checking that the challenge they provided, s, is the xor of the challenges in the respective conversations — $s = e_0 \oplus e_1$.

$$\begin{aligned} &\mathit{check}_{OR}((x_0, x_1), (a_0, a_1), s, ((e_0, z_0), (e_1, z_1))) \\ &\qquad = (s = e_0 \oplus e_1 \wedge e_0 \in \mathit{challenge\text{-}space} \wedge e_1 \in \mathit{challenge\text{-}space} \\ &\qquad\qquad \wedge \mathit{check}_0(x_0, a_0, e_0, z_0) \wedge \mathit{check}_1(x_1, a_1, e_1, z_1)) \end{aligned} \tag{4.17}$$

The *challenge-space* is defined as the carrier set of L — $\mathit{challenge\text{-}space}_{OR} = \mathit{carrier}(L)$ and the public input (x_0, x_1) is valid if x_i is a valid public input with respect to its underlying Σ-protocol, that is:

$$\mathit{valid\text{-}pub}_{OR} = \{(x_0, x_1).\ x_0 \in \mathit{valid\text{-}pub}_0 \wedge x_1 \in \mathit{valid\text{-}pub}_1\} \tag{4.18}$$

We import the *Σ-protocol-base* locale — under the name *Σ-OR* — so we can reason about the properties of Σ-protocols. First we show completeness.

The proof of the completeness property requires Condition 2 of the HVZK definition in Definition 3. It is required because the simulated transcript in the OR protocol must also produce a valid conversation if the Verifier is to accept the proof, without Condition 2 we have no guarantee that this is the case.

Lemma 1. *(in Σ-OR-proof)* **shows** *Σ-OR.completeness*

Proof. For ease we split the proof into cases depending on which relation holds. For the case where $Rel_1(x_1, w)$ holds the components corresponding to Rel_1 are generated using the Σ-protocol Σ_1, whereas the components corresponding to Rel_0 are simulated using S_0. For the correctly generated case (Rel_1) the check outputs true due to the completeness property of Σ_1. For the simulated case (Rel_0) we use the HVZK property (Condition 2) from Σ_0 to show the check outputs true. □

To prove HVZK we use the following simulator. This is constructed by defining $S_{raw,OR}$ in the first instance, we only give the unfolded full definition of the simulator here.

$$\begin{aligned} &\Sigma\text{-}OR.S_{OR}((x_0, x_1), s) = do\ \{ \\ &\quad e_1 \leftarrow \mathit{uniform}(\mathit{carrier}(L)); \\ &\quad (a_1, e_1', z_1) \leftarrow S_1(x_1, e_1); \\ &\quad \mathit{let}\ e_0 = s \oplus e_1; \\ &\quad (a_0, e_0', z_0) \leftarrow S_0(x_0, e_0); \\ &\quad \mathit{let}\ z = ((e_0', z_0), (e_1', z_1)); \\ &\quad \mathit{return}((a_0, a_1), s, z)\} \end{aligned} \tag{4.19}$$

Note, in constructing the simulator we had a design choice: sample either e_1 or e_0 and construct the other — either choice results in the same simulator.

Lemma 2. *(in Σ-OR-proof)* **shows** *Σ-OR.HVZK*

Proof. We simulate the real view by running the simulator (given in Equation 4.19) for both relations. The challenges we give to the simulators (e_0 and e_1) are related by $s = e_0 \oplus e_1$, where we sample e_1 uniformly (we could have sampled e_0) and s is the challenge in the OR construction. This asymmetry (we must sample one of e_0 or e_1) is dealt with using the lemma given in Equation 4.9. In the case where $Rel_0(x_0, w)$ holds the result comes directly by writing the components from Σ_0 in Σ-$OR.R$ into the real view then using the HZVK property of Σ_0 to rewrite the real view as the simulator. In the case where $Rel_1(x_1, w)$ holds we follow the same process but use Equation 4.9 in the last step. □

To construct the special soundness adversary we condition on the case $e_0 \neq e'_0$. The reason for this is that in the proof of the special soundness property we show that either $e_0 \neq e'_0$ or $e_1 \neq e'_1$ must hold (depending on which relation to witness pertains to). In either case the adversary outputs the witness to the respective relation using the special soundness adversaries from Σ_0 or Σ_1.

```
$\mathcal{A}_{ss,OR}((x_0, x_1), conv, conv') = do \{$
  $let\ ((a_0, a_1), s, (e_0, z_0), e_1, z_1) = conv;$
  $let\ ((a_0, a_1), s', (e'_0, z'_0), e'_1, z'_1) = conv';$
  $if\ (e_0 \neq e'_0)\ then\ do\ \{$
    $w_0 \leftarrow \mathcal{A}_{ss,0}(x_0, (a_0, e_0, z_0), (a_0, e'_0, z'_0));$          (4.20)
    $return(Inl(w_0))\}$
  $else\ do\ \{$
    $w_1 \leftarrow \mathcal{A}_{ss,1}(x_1, (a_1, e_1, z_1), (a_1, e'_1, z'_1));$
    $return(Inr(w_1))\}\ \}$
```

Lemma 3. *(in Σ-OR-proof)* **shows** Σ-*OR.special-soundness*

Proof. We must show $\mathcal{A}_{ss,OR}$ is lossless and always outputs a witness for Rel_{OR}. We have two conversations $((a_0, a_1), s, (e_0, z_0), (e_1, z_1))$ and $((a_0, a_1), s', (e'_0, z'_0), (e'_1, z'_1))$ on public inputs x_0 and x_1 respectively. We can assume the following hold (the assumptions in the statement of special soundness):

- $s \neq s'$
- $check_{OR}((x_0, x_1), (a_0, a_1), s, (e_0, z_0), (e_1, z_1))$
- $check_{OR}((x_0, x_1), (a_0, a_1), s', (e'_0, z'_0), (e'_1, z'_1))$
- $(x_0, x_1) \in valid\text{-}pub_{OR}$
- $s, s' \in challenge\text{-}space_{OR}$

From $s \neq s'$ we show that $e_0 \neq e'_0 \vee e_1 \neq e'_1$ and partition the proof on the case $e_0 \neq e'_0$. When this condition holds we know the conditions for the special soundness property for Σ_0 hold and thus $\mathcal{A}_{ss,0}$ is lossless and outputs a witness to Rel_0. The branch of the if statement that is invoked in $\mathcal{A}_{ss,OR}$ in this case calls $\mathcal{A}_{ss,0}$ and therefore outputs a witness to Rel_0. The proof for the second case, $e_1 \neq e'_1$, is analogous. □

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier}\\
(x_0,x_1), w & & (x_0,x_1)\\
(r_b,a_b) \xleftarrow{\$} init_b(x_b,w) & & \\
e_{1-b} \xleftarrow{\$} L & & \\
(a_{1-b},e_{1-b},z_{1-b}) \xleftarrow{\$} S_b(x_{1-b},e_{1-b}) & \xrightarrow{\text{initial msg: } (a_0,a_1)} & (a_0,a_1)\\
s & \xleftarrow{\text{challenge: } s} & s \xleftarrow{\$} L\\
e_b = s \oplus e_{1-b} & & \\
z_b \leftarrow response_b(r_b,w,e_b) & \xrightarrow{\text{response: } (e_0,z_0,e_1,z_1)} & (e_0,z_0,e_1,z_1)\\
 & & check_0(x_0,a_0,e_0,z_0)\\
 & & check_1(x_1,a_1,e_1,z_1)
\end{array}$$

Figure 4.3: Change to be AND construction

Using Lemmas 1, 2 and 3 we prove the OR construction is a Σ-protocol.

Theorem 7. *(in Σ-OR-proof)* **shows** *OR-Σ.Σ-protocol*

4.3.1.2 *AND construction for Σ-protocols*

Section 4.3.1.1 showed how a Σ-protocol for the OR of two relations can be constructed. Here we show how this can be done for the AND of two relations.

The relation Rel_{AND} is defined as:

$$Rel_{AND} = \{((x_0,x_1),(w_0,w_1)).\ ((x_0,w_0) \in Rel_0 \wedge (x_1,w_1) \in Rel_1)\}. \tag{4.21}$$

The construction of the initial message shows the parallel execution in both sides:

$$\begin{aligned}
&init_{AND}((x_0,x_1),(w_0,w_1)) = do\ \{\\
&\quad (r_0,a_0) \leftarrow init_0(x_0,w_0);\\
&\quad (r_1,a_1) \leftarrow init_1(x_1,w_1);\\
&\quad return((r_0,r_1),(a_0,a_1))\}
\end{aligned} \tag{4.22}$$

The Prover responds with responses to the challenge for each witness also and the check function requires that both conversations are valid as shown below.

$$\begin{aligned}
&response_{AND}((r_0,r_1),(w_0,w_1),s) = do\ \{\\
&\quad z_0 \leftarrow response_0(r_0,w_0,s);\\
&\quad z_1 \leftarrow response_1(r_1,w_1,s);\\
&\quad return(z_0,z_1)\}
\end{aligned} \tag{4.23}$$

$$
\begin{aligned}
&check_{AND}((x_0, x_1), (a_0, a_1), s, (z_0, z_1)) = \\
&\quad (check_0(x_0, a_0, s, z_0) \wedge check_1(x_1, a_1, s, z_1))
\end{aligned}
\tag{4.24}
$$

Analogous to the case of the OR construction we import the Σ-protocol locale as Σ-*AND*. The proofs are able to directly use the corresponding properties of Σ_0 and Σ_1. We first show completeness.

Lemma 4. *(in Σ-AND)* **shows** Σ-*AND.completeness*

Proof. The executions of Σ_0 and Σ_1 are run in parallel, therefore the completeness properties of Σ_0 and Σ_1 can be applied straightforwardly for completeness to be realised. □

For the HVZK property we construct the following simulator, as usual we give the unfolded version here for simplicity.

$$
\begin{aligned}
&S_{AND}((x_0, x_1), e) = do\ \{ \\
&\quad (a_0, c_0, z_0) \leftarrow S_0(x_0, e); \\
&\quad (a_1, c_1, z_1) \leftarrow S_1(x_1, e); \\
&\quad return((a_0, a_1), e, (z_0, z_1))\}
\end{aligned}
\tag{4.25}
$$

Lemma 5. *(in Σ-AND)* **shows** Σ-*AND.HVZK*

Proof. The conversations for the AND construction are the conversations for Σ_0 and Σ_1 combined, thus both can be simulated by the HVZK property of Σ_0 and Σ_1, the simulator (given in Equation 4.25) does exactly this. □

The adversary we construct to prove the special soundness property is as follows.

$$
\begin{aligned}
&\mathcal{A}_{ss,AND}((x_0, x_1), conv, conv') = do\ \{ \\
&\quad let\ ((a_0, a_1), e, (z_0, z_1)) = conv; \\
&\quad let\ ((a'_0, a'_1), e', (z'_0, z'_1)) = conv'; \\
&\quad w_0 \leftarrow \mathcal{A}_{ss,0}(x_0, (a_0, e, z_0), (a'_0, e', z'_0)); \\
&\quad w_1 \leftarrow \mathcal{A}_{ss,1}(x_1, (a_1, e, z_1), (a'_1, e', z'_1)); \\
&\quad return(w_0, w_1)\}
\end{aligned}
\tag{4.26}
$$

Lemma 6. *(in Σ-AND)* **shows** Σ-*AND.special-soundness*

Proof. The special soundness adversary, $\mathcal{A}_{ss,AND}$, runs the special soundness adversaries for both Σ_0 and Σ_1 to get the witnesses for each relation. The correct witnesses are outputted due to the adversaries for Σ_0 and Σ_1 outputting the correct witnesses for their respective protocols and $\mathcal{A}_{ss,AND}$ is lossless as the adversaries it uses are lossless, again due to the special soundness soundness property of Σ_0 and Σ_1. □

Combining the properties we can show the construction is a Σ-protocol.

Theorem 8. *(in Σ-AND)* **shows** Σ-*AND*.Σ-*protocol*

$$\begin{tikzcd}
\text{Prover} & & \text{Verifier} \\
(h,w) & & h \\
r \overset{\$}{\leftarrow} \mathbb{Z}_{|G|} & & \\
a \leftarrow g^r \arrow[rr, "\text{initial msg: } a"] & & a \\
& & e \overset{\$}{\leftarrow} \mathbb{Z}_{|G|} \arrow[ll, "\text{challenge: } e"'] \\
z \leftarrow (w \cdot e + r) \ mod \ |G| \arrow[rr, "\text{response: } z"] & & z \\
& & \text{check: } a \cdot h^e = g^z \\
& & \text{accepts or rejects}
\end{tikzcd}$$

Figure 4.4: The Schnorr Σ-protocol.

4.3.2 *The Schnorr Σ-protocol*

The Schnorr protocol uses a cyclic group G with generator g and considers the discrete log relation which on public input h requires the witness to be the discrete log of h in G — $h = g^w$. The Schnorr Σ-protocol is given in Figure 4.4. We explain the protocol Example 1 and provide a detailed paper proof of its security in Theorem 1 in section 2.2.

The most interesting part of the formal proof is special soundness. The adversary must output $w = (\frac{z-z'}{e-e'}) \ mod \ |G|$. To formalise this part in Isabelle we were required to formalise the inverses in a field; more specifically, as we just work modulo the size of the field, we need to formalise what it means to be an inverse modulo the size of the field. We take a small aside here to discuss how we do this as it is relevant for many of the remaining proofs in this section.

4.3.2.1 *Formalising Inverses*

Throughout our formalisation we work with natural numbers instead of formalising a field construction. Therefore we work modulo q whenever we actually work in a field. One issue we encounter is constructing inverses modulo q. We are required to reason about the inverses of elements in a field in many places in our formalisation, for example the special soundness adversary outputs $w = (\frac{z-z'}{e-e'}) \ mod \ |G|$ in the Schnorr protocol. We show how we formalise such an inverse.

Obviously, the standard division function on natural numbers is not suitable to obtain an inverse in the field modulo q. Instead, we use the existing number theory formalisation in Isabelle's standard library, in particular Bezout's function (*bezw*). Bezout's identity informally says: let a and b be integers such that $gcd(a,b) = d$ then there exist integers x and y such that $a \cdot x + b \cdot y = d$. In Isabelle, the function $bezw(a,b)$ returns the pair (x,y) of witnesses to Bezout's identity. So we obtain the inverse of a as $fst(bezw(a,q))$. For readability we define an abbreviation for the inverse.

$$inv_q(a) = fst(bezw(a,q))$$

We prove the following general lemma, which we find is sufficient in all the cases where reasoning about the inverse is required in our formalisation.

Lemma 7. **assumes** $gcd(a, q) = 1$
shows $[a \cdot inv_q(a) = 1] \ mod \ q$

Proof. The function *bezw* outputs a pair of witnesses to Bezout's identity, using this along with the assumption that $gcd(a, q) = 1$ we have

$$inv_q(a) \cdot a + snd(bezw(a, q)) \cdot q = 1$$

Considering this modulo q the result comes easily as the second term on the left hand side vanishes. □

4.3.2.2 *Proving the Schnorr Σ-protocol is a Sigma-protocol*

In the case of the Schnorr Σ-protocol we instantiate q as $|G|$. The assumption, in general, holds in our usage as $a < |G|$, $a \neq 0$ and $|G|$ is prime.

The Schnorr Σ-protocol is defined over a cyclic group of prime order. We use the construction of cyclic groups from [51] to fix a group $\mathcal{G}$ in the locale we work in as follows.

$$\begin{aligned} &\textbf{locale } \textit{schnorr-base} = \\ &\quad \textbf{fixes } \mathcal{G} \ :: \ \textit{'grp cyclic-group} \\ &\quad \textbf{assumes } \textit{prime}(\textit{order}(\mathcal{G})) \end{aligned} \tag{4.27}$$

To show the Schnorr Σ-protocol has the desired properties of Σ-protocols we explicitly define the parameters required in our Σ-protocol framework. We define

$$init^S, response^S, check^S, R^S_{DL}, S^S_{raw}, \mathcal{A}^S_{ss}, challenge\text{-}space^S, valid\text{-}pub^S$$

where the superscript S denotes that these constants are for the Schnorr Σ-protocol. We make these definitions inside the context of the locale. The types of the components of the protocol are made more concrete from the definitional theory of Σ-protocols, in particular we define the following type synonyms.

type-synonym *witness* = *nat*
type-synonym *'grp pub-in* = *'grp*
type-synonym *'grp msg* = *'grp*
type-synonym *rand* = *nat*
type-synonym *challenge* = *nat*
type-synonym *response* = *nat*

For the Schnorr Σ-protocol the relation is the discrete log relation, as given informally in Equation 2.1; formally this is encoded into Isabelle as

$$R^S_{DL} = \{(h, w).\ h = g^w\}. \tag{4.28}$$

The programs $init^S$, $response^S$ and $check^S$ correspond to the stages of the protocol given in Figure 4.4.

$$\begin{aligned}&init^S \ :: \ (\text{'}grp\ pub\text{-}in \times witness) \Rightarrow (rand \times \text{'}grp\ msg)\ spmf\\ &init^S(h, w) = do\ \{\\ &\quad r \leftarrow samp\text{-}uniform(|G|);\\ &\quad return(r, g^r)\}\end{aligned} \tag{4.29}$$

$$\begin{aligned}&response^S \ :: \ rand \Rightarrow witness \Rightarrow challenge \Rightarrow response\ spmf\\ &response^S(r, w, e) = return((w \cdot c + r)\ mod\ |G|)\end{aligned} \tag{4.30}$$

$$\begin{aligned}&check^S \ :: \ \text{'}grp\ pub\text{-}in \Rightarrow \text{'}grp\ msg \Rightarrow challenge \Rightarrow response \Rightarrow bool\\ &check^S(h, a, e, z) = (a \otimes h^e = g^z)\end{aligned} \tag{4.31}$$

A public input is valid if it is in the group, $valid\text{-}pub^S = carrier(G)$. And the challenge set is the set of naturals up to the order of G, $challenge\text{-}space^S = \{0, \ldots, |G|\}$.

We show these constants are an instantiation of the Σ-*protocol-base* locale (Equation 4.4). As explained in section 3.3.1.1 we do this using the sublocale command; this is an extension of the sublocale given in Equation 4.1.

sublocale *Schnorr-Σ* : Σ-*protocol-base* $init^S$ $response^S$ $check^S$
R^S_{DL} S^S_{raw} $\mathcal{A}^S_{ss}$ $challenge\text{-}space^S$ $valid\text{-}pub^S$

We also inherit the cyclic group properties of the group G by forming the following locale.

$$\textbf{locale}\ schnorr = schnorr\text{-}base + cyclic\text{-}group(G) \tag{4.32}$$

In this context we can prove the desired properties of the Schnorr Σ-protocol.

Lemma 8. *(in schnorr)* **shows** *Schnorr-Σ.completeness*

Proof. Completeness follows after proving the identity $g^r \otimes (g^w)^e = g^{r+w\cdot e}$ and passing it as a rewrite rule to the simplifier.

□

Second we consider special soundness. To prove this property we construct an adversary that can extract the witness from accepting conversations of the protocol. We informally gave the construction of this adversary in the previous section; given two accepting conversations (a, e, z) and (a, e', z') the adversary outputs $(\frac{z-z'}{e-e'})\ mod\ |G|$. When encoding of the adversary in Isabelle we must be mindful of whether $e > e'$; as we are working with naturals bounded subtraction in the denominator $e - e'$ will return 0 if $e < e'$ (we know that $e \neq e'$ as it is a condition on the conversations given to the adversary).

$$
\begin{aligned}
&\mathcal{A}^S_{ss}(h, c_1, c_2) = do\ \{ \\
&\quad let\ (a, e, z) = c_1; \\
&\quad let\ (a', e', z') = c_2; \\
&\quad return(if\ e > e'\ then\ (z - z') \cdot inv_G(e - e')\ mod\ |G| \\
&\qquad\qquad else\ (z' - z) \cdot inv_G(e' - e)\ mod\ |G|)\}
\end{aligned}
\tag{4.33}
$$

Using this adversary we prove the special soundness property for the Schnorr Σ-protocol.

Lemma 9. *(in schnorr)* **shows** *Schnorr-Σ.special-soundness*

Proof. The adversary $\mathcal{A}^S_{ss}$ is clearly lossless — it does not do any probabilistic sampling. Showing the adversary outputs a witness to the relation is proven by using Lemma 7 to rewrite the output of the adversary in a similar manner to the paper proof given in section 4.3.2. □

Finally we consider the Honest Verifier Zero Knowledge property. This proof technique follows the technique of simulation-based proofs that was formally introduced in Isabelle and CryptHOL in [21]. To prove HVZK we define the simulator, S^S_{raw}, which in turn defines *Schnorr-Σ.S*^{S}. We then prove this mimics the real view. The unfolded simulator is formed as follows; recall the intuition of sampling the response first and constructing the initial message from it.

$$
\begin{aligned}
&Schnorr\text{-}\Sigma.S^S(h, e) = do\ \{ \\
&\quad z \leftarrow samp\text{-}uniform(|G|); \\
&\quad let\ a = g^z \otimes (h^e)^{-1}; \\
&\quad return\ (a, e, z)\}
\end{aligned}
\tag{4.34}
$$

Lemma 10. *(in schnorr)* **shows** *Schnorr-Σ.HVZK*(h, w)

Proof. First we show the simulator and the real view are equal. The unfolded real view can be written as:

$$
\begin{aligned}
&Schnorr\text{-}\Sigma.real\text{-}view^S(h, w) = do\ \{ \\
&\quad r \leftarrow samp\text{-}uniform(|G|); \\
&\quad let\ (r, a) = (r, g^r); \\
&\quad c \leftarrow samp\text{-}uniform(|G|); \\
&\quad let\ z = (w \cdot c + r)\ mod\ |G|; \\
&\quad return\ (a, c, z)\}
\end{aligned}
\tag{4.35}
$$

The juxt of the proof is showing that z constructed in the real view is a uniform sample — as it is in the simulator — this destroys any information passed to V about the witness. To do this we use the following one time pad lemma:

$$map(\lambda b.\ (y + b)\ mod\ q,\ samp\text{-}uniform(q)) = samp\text{-}uniform(q)$$

$$\begin{array}{ccc}
\text{Prover} & & \text{Verifier}\\
((h_0,h_1),w) & & (h_0,h_1)\\
r \xleftarrow{\$} \mathbb{Z}_{|G|} & & \\
a_0 \leftarrow g^r,\ a_1 \leftarrow g'^r & \xrightarrow{\text{initial msg: } (a_0,a_1)} & a\\
 & \xleftarrow{\text{challenge: } e} & e \xleftarrow{\$} \mathbb{Z}_{|G|}\\
z \leftarrow (w \cdot e + r)\ mod\ |G| & \xrightarrow{\text{response: } z} & z\\
 & & \text{check: } a_0 \cdot h_0^e = g^z \text{ and } a_1 \cdot h_1^e = g'^z
\end{array}$$

Figure 4.5: The Chaum-Pedersen Σ-protocol.

To use this lemma in the proof we must rewrite some of the terms in the real view. These rewriting statements of equality are nearly always needed when using such lemmas as the remaining probabilistic program can no longer depend on b and must be rewritten in terms of the other variables.

Second we show the output of the simulator is a valid transcript. This part of the proof comes easily and in a similar manner to the proof of correctness. □

Using Lemmas 8, 9 and 10 we show that the Schnorr Σ-protocol is in fact a Σ-protocol.

Theorem 9. *(in schnorr)* **shows** *Schnorr-Σ.Σ-protocol*

4.3.3 *Chaum-Pedersen Σ-protocol*

In this section we detail our formalisation of the Chaum-Pedersen Σ-protocol [27]. The protocol is run over a cyclic group G of prime order where g and g' are generators of G. The relation considered here is often described as the equality of discrete logs relation.

$$Rel_{CP} = \{((h_0,h_1),w).\ h_0 = g^w \wedge h_1 = g'^w\} \tag{4.36}$$

The protocol is shown in Figure 4.5.

In the locale *chaum-ped-Σ-base* we fix the group G and a natural x that we use to construct $g' = g^x$.

$$\begin{array}{l}
\textbf{locale}\ \mathit{chaum\text{-}ped\text{-}\Sigma\text{-}base} = \\
\quad \textbf{fixes}\ G\ ::\ \mathit{'grp\ cyclic\text{-}group} \\
\qquad \textbf{and}\ x\ ::\ \mathit{nat} \\
\quad \textbf{assumes}\ \mathit{prime}(|G|)
\end{array} \tag{4.37}$$

As usual we define the components of the Σ-protocol.

$$\begin{aligned}&\mathit{init}_{CP}((h_0,h_1),w) = \mathit{do}\ \{\\ &\quad r \leftarrow \mathit{samp\text{-}uniform}(|G|);\\ &\quad \mathit{return}(r,(g^r,g'^r))\}\end{aligned} \tag{4.38}$$

$$\begin{aligned}&\mathit{check}_{CP}((h_0,h_1),(a_0,a_1),e,z)\\ &\quad = (a_0 \otimes h_0^e = g^z \wedge a_1 \otimes h_1^e = g'^z)\end{aligned} \tag{4.39}$$

$$\mathit{response}_{CP}(r,w,e) = (\mathit{return}(w \cdot e + r)\ \mathit{mod}\ |G|) \tag{4.40}$$

After importing the Σ-*protocol-base* locale as *CP*-Σ we construct a new locale where we import the cyclic group properties of G in which to prove the properties of the protocol.

$$\textbf{locale}\ \mathit{chaum\text{-}ped\text{-}\Sigma} = \mathit{chaum\text{-}ped\text{-}\Sigma\text{-}base} + \mathit{cyclic\text{-}group}(G) \tag{4.41}$$

The proofs of the properties for the Chaum-Pedersen Σ-protocol are similar to the proofs of the Schnorr Σ-protocol (Lemmas 8, 9 and 10) the general difference being we do everything twice as we have two initial messages sent compared to one in the Schnorr protocol. The statements of the security properties are given below.

Lemma 11. *(in chaum-ped-Σ)* **shows** *CP-Σ.completeness*

The unfolded simulator for the HVZK property is shown below. When compared to the Schnorr simulator we see this simulator creates two initial messages as opposed to one. The intuition behind the construction of the simulator is to uniformly sample the response to ensure it contains no information about the witness (by definition). The other components of the output can then be constructed around this uniform sample.

$$\begin{aligned}&S_{CP}((h_0,h_1),e) = \mathit{do}\ \{\\ &\quad z \leftarrow \mathit{samp\text{-}uniform}(|G|);\\ &\quad \mathit{let}\ a = g^z \otimes (h_0^{-e});\\ &\quad \mathit{let}\ a' = g'^z \otimes (h_1^{-e});\\ &\quad \mathit{return}((a,a',e,z))\}\end{aligned} \tag{4.42}$$

Lemma 12. *(in chaum-ped-Σ)* **shows** *CP-Σ.HVZK*

For the special soundness property we construct the following adversary.

$$\begin{aligned}&\mathcal{A}_{ss,CP}((h_0,h_1),c_1,c_2) = \mathit{do}\ \{\\ &\quad \mathit{let}\ ((a,a'),e,z) = c_1;\\ &\quad \mathit{let}\ ((b,b'),e',z') = c_2;\\ &\quad \mathit{return}(\mathit{if}\ e > e'\ \mathit{then}\ (z - z') \cdot \mathit{inv}_G(e - e')\\ &\qquad\qquad \mathit{else}\ (z' - z) \cdot \mathit{inv}_G(e' - e))\}\end{aligned} \tag{4.43}$$

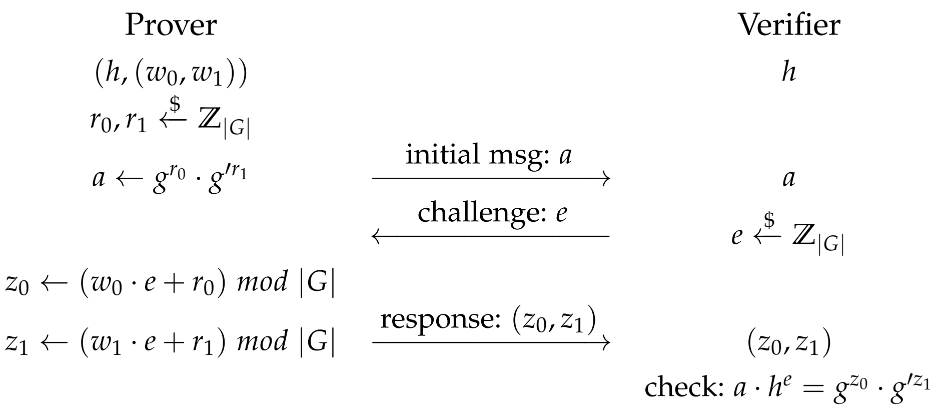

Prover
Verifier
$(h, (w_0, w_1))$
h
$r_0, r_1 \xleftarrow{\$} \mathbb{Z}_{|G|}$
$a \leftarrow g^{r_0} \cdot g'^{r_1}$
initial msg: a
a
challenge: e
$e \xleftarrow{\$} \mathbb{Z}_{|G|}$
$z_0 \leftarrow (w_0 \cdot e + r_0)\ mod\ |G|$
$z_1 \leftarrow (w_1 \cdot e + r_1)\ mod\ |G|$
response: (z_0, z_1)
(z_0, z_1)
check: $a \cdot h^e = g^{z_0} \cdot g'^{z_1}$

Figure 4.6: The Okamoto Σ-protocol.

Lemma 13. *(in chaum-ped-Σ)* **shows** *CP-Σ.special-soundness*

Together Lemmas 12, 13 and 11 imply our formalisation of the Chaum-Pedersen Σ-protocol is a Σ-protocol.

Theorem 10. *(in chaum-ped-Σ)* **shows** *CP-Σ.Σ-protocol*

4.3.4 *Okamoto Σ-protocol*

In this section we detail our formalisation of the Okamoto Σ-protocol [27]. The protocol is run over a cyclic group G of prime order where g and g' are generators of G. The relation is as follows.

$$Rel_{Ok} = \{(h, (w_0, w_1)).\ h = g^{w_0} \otimes g'^{w_1}\} \tag{4.44}$$

The protocol is shown in Figure 4.6.

In the locale *okamoto-Σ-base* we fix the group G and a natural x that we use to construct $g' = g^x$, this is equivalent to the Chaum-Pedersen Σ-protocol.

$$\begin{aligned}&\textbf{locale}\ \mathit{okamoto\text{-}\Sigma\text{-}base} =\\ &\quad \textbf{fixes}\ G\ ::\ \text{'}\mathit{grp\ cyclic\text{-}group}\\ &\qquad \textbf{and}\ x\ ::\ \mathit{nat}\\ &\quad \textbf{assumes}\ \mathit{prime}(|G|)\end{aligned} \tag{4.45}$$

As usual we define the components of the Σ-protocol in turn.

$$\begin{aligned}&\mathit{init}_{Ok}(h, w) = \mathit{do}\ \{\\ &\quad r_0 \leftarrow \mathit{samp\text{-}uniform}(|G|);\\ &\quad r_1 \leftarrow \mathit{samp\text{-}uniform}(|G|);\\ &\quad \mathit{return}((r_0, r_1), (g^{r_0} \otimes g'^{r_1}))\}\end{aligned} \tag{4.46}$$

$$\begin{aligned}&\mathit{response}_{\mathit{Ok}}((r_0, r_1), (w_0, w_1), e) = \\ &\quad \mathit{return}((w_0 \cdot e + r_0)\ \mathit{mod}\ |G|, (w_1 \cdot e + r_1)\ \mathit{mod}\ |G|)\end{aligned} \tag{4.47}$$

$$\mathit{check}_{\mathit{Ok}}(h, a, e, (z_0, z_1)) = (a \otimes h^e = g^{z_0} \otimes g'^{z_1}) \tag{4.48}$$

After importing the Σ-*protocol-base* locale as *O*-Σ we construct a new locale where we import the cyclic group properties of *G* in which to prove the properties of the protocol.

$$\textbf{locale}\ \mathit{okamoto}\text{-}\Sigma = \mathit{okamoto}\text{-}\Sigma\text{-}\mathit{base} + \mathit{cyclic}\text{-}\mathit{group}(G) \tag{4.49}$$

The proofs are again of a similar flavour to those of the Schnorr and Chaum-Pedersen Σ-protocols, therefore we do not elaborate on them here.

Lemma 14. *(in okamoto-Σ)* **shows** *O-Σ.completeness*

The unfolded simulator used to prove the HVZK property is shown below.

$$\begin{aligned}&S_{\mathit{Ok}}(h, e) = \mathit{do}\ \{ \\ &\quad z_0 \leftarrow \mathit{samp}\text{-}\mathit{uniform}(|G|); \\ &\quad z_1 \leftarrow \mathit{samp}\text{-}\mathit{uniform}(|G|); \\ &\quad \mathit{let}\ a = g^{z_0} \otimes g'^{z_1} \otimes (h^{-e}); \\ &\quad \mathit{return}(a, e, (z_0, z_1))\}\end{aligned} \tag{4.50}$$

Lemma 15. *(in okamoto-Σ)* **shows** *O-Σ.HVZK*

To prove the special soundness property we use the following adversary. We note the only difference to the previous adversaries we have seen is that the return statement outputs a 2-tuple.

$$\begin{aligned}&\mathcal{A}_{\mathit{ss,Ok}}(h, c_1, c_2) = \mathit{do}\ \{ \\ &\quad \mathit{let}\ (a, e, (z_0, z_1)) = c_1; \\ &\quad \mathit{let}\ (a', e', (z_0', z_1')) = c_2; \\ &\quad \mathit{return}(\mathit{if}\ e > e'\ \mathit{then}\ (z_0 - z_0') \cdot \mathit{inv}_G(e - e') \\ &\qquad\qquad \mathit{else}\ (z_0' - z_0) \cdot \mathit{inv}_G(e' - e), \\ &\qquad\qquad\qquad \mathit{if}\ e > e'\ \mathit{then}\ (z_1 - z_1') \cdot \mathit{inv}_G(e - e') \\ &\qquad\qquad\qquad\quad \mathit{else}\ (z_1' - z_1) \cdot \mathit{inv}_G(e' - e))\}\end{aligned} \tag{4.51}$$

Lemma 16. *(in okamoto-Σ)* **shows** *O-Σ.special-soundness*

Together Lemmas 15, 16 and 14 imply our formalisation of the Okamoto Σ-protocol is a Σ-protocol.

Theorem 11. *(in okamoto-Σ)* **shows** *O-Σ.Σ-protocol*

In this section we have shown how we formally define Σ-protocols and their security properties and how we instantiate our framework for reasoning about well known protocols and constructions. The compound constructions we consider are proved at a general level; if desired one could instantiate the Schnorr, Chaum-Pedersen or Okamoto Σ-protocols for this proof in a matter of lines.

5 COMMITMENT SCHEMES

5.1 INTRODUCTION

In this Chapter we introduce our framework for reasoning about commitment schemes. Like the other definitional frameworks in this thesis we are able to instantiate commitment schemes and prove they are secure relative to our definitions.

Commitment schemes are a cryptographic primitive, run between a Committer C and a Verifier V, that allow the Committer to commit to a chosen message, while keeping it private, and at a later time reveal the message that was committed to. We point the reader back to Section 2.3 for our introduction to commitment schemes or to [72] for a more thorough exposition.

CHAPTER OUTLINE Figure 5.1 outlines the work flow in this chapter. In Section 5.2 we introduce our framework for reasoning about commitment schemes and then instantiate it in Section 5.3 for the Rivest commitment scheme. We include the Pedersen commitment scheme in this Figure as in our formalisation we prove it secure from scratch. We also present the proof obtained from the general result, the construction from Σ-protocols, in the next chapter — our formalised proof of the Pedersen commitment scheme from scratch is given for comparison of proof effort with the result in the next chapter.

The work in this chapter has been published in [22, 24, 25].

5.2 FORMALISING COMMITMENT SCHEMES

We formalise commitment schemes analogously to Σ-protocols. First we fix the required parameters (as functions) in the locale, *commit-base*, given in 5.1.

$$\begin{aligned}
&\textbf{locale}\ \textit{commit-base} = \\
&\quad \textbf{fixes}\ \textit{key-gen} :: (\text{'}ck \times \text{'}vk)\ \textit{spmf} \\
&\qquad \textbf{and}\ \textit{commit} :: \text{'}ck \Rightarrow \text{'}plain \Rightarrow (\text{'}com \times \text{'}open)\ \textit{spmf} \\
&\qquad \textbf{and}\ \textit{verify} :: \text{'}vk \Rightarrow \text{'}plain \Rightarrow \text{'}com \Rightarrow \text{'}open \Rightarrow \textit{bool spmf} \\
&\qquad \textbf{and}\ \textit{valid-msg} :: \text{'}plain \Rightarrow \textit{bool}
\end{aligned} \tag{5.1}$$

The probabilistic programs *key-gen*, *commit* and *verify* will correspond to the three components of a commitment scheme. The key generation function outputs the keys that are available to the Committer and Verifier. If, for example, all the keys are shared then we have $ck = vk$. The predicate *valid-msg* ensures the messages outputted by the adversary in the hiding game are valid, for example we may require them to be group elements.

Using these fixed parameters we define the correctness, hiding and binding for commitment schemes.

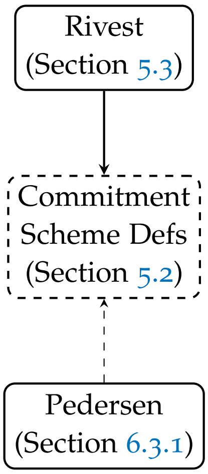


Figure 5.1: Outline of the formalisation of commitment schemes in this thesis.

For the correctness property we define the probabilistic program *correct-game*.

$$\begin{aligned}
&\textit{correct-game}(m) = \textit{do} \ \{\\
&\quad (ck, vk) \leftarrow \textit{key-gen};\\
&\quad (c, d) \leftarrow \textit{commit}(ck, m);\\
&\quad \textit{return}(\textit{verify}(vk, m, c, d))\}
\end{aligned} \tag{5.2}$$

For a commitment scheme to be correct we require that for all valid messages *correct-game* always returns True.

Definition 5. *(Correctness)*

$$\textit{correct} = (\forall m.\ \textit{valid-msg}(m) \longrightarrow \mathcal{P}[\textit{correct-game}(m) = \textit{True}] = 1)$$

When considering the hiding and binding properties we define the advantage an adversary has of winning the corresponding security game, or in the best cases, perfect hiding or binding.

The hiding game, *hiding-game* is defined as follows.

$$\begin{aligned}
&\textit{hiding-game}\ (\mathcal{A}_1, \mathcal{A}_2) = \textit{TRY do}\ \{\\
&\quad (ck, vk) \leftarrow \textit{key-gen};\\
&\quad ((m_0, m_1), \sigma) \leftarrow \mathcal{A}_1(vk);\\
&\quad \_ \leftarrow \textit{assert}(\textit{valid-msg}(m_0) \wedge \textit{valid-msg}(m_1));\\
&\quad b \leftarrow \textit{coin};\\
&\quad (c, d) \leftarrow \textit{commit}(ck, (\textit{if}\ b\ \textit{then}\ m_1\ \textit{else}\ m_2));\\
&\quad b' \leftarrow \mathcal{A}_2(c, \sigma);\\
&\quad \textit{return}(b = b')\}\ \textit{ELSE coin}
\end{aligned} \tag{5.3}$$

In this game the challenger asks the adversary to output two messages, commits one of the messages and hands it back to the adversary who must determine which message was committed. The adversary is said to win the game if it guesses correctly. Formally the adversary is split into two parts $(\mathcal{A}_1, \mathcal{A}_2)$, the first part outputs the messages and the second its guess at which messages was committed to. We highlight that we must check the messages (m_0, m_1) outputted by the adversary are valid, if the assertion fails then the *ELSE* branch is invoked and the adversary only wins the game half the time (equivalent to if it guessed randomly). Also note the two parts of the adversary must be allowed to pass state to each other. The hiding advantage is defined with respect to the hiding game.

Definition 6. *(Hiding Advantage)*

$$\textit{hiding-advantage}(\mathcal{A}) = |\mathcal{P}[\textit{hiding-game}(\mathcal{A}) = \textit{True}] - \frac{1}{2}|$$

Definition 7. *(Perfect Hiding)*

$$\textit{perfect-hiding}(\mathcal{A}) = (\textit{hiding-advantage}(\mathcal{A}) = 0)$$

The binding game asks the adversary (now acting as the Committer) to output a commitment c and two pairs of messages and opening values $((m, d), (m', d'))$ such that they both verify — the messages outputted by the adversary must be distinct and valid, with respect to c, which is accounted for by the assert statement.

$$\begin{aligned}
&\textit{binding-game } \mathcal{A} = \ \textit{TRY do} \ \{\\
&\quad (ck, vk) \leftarrow \textit{key-gen};\\
&\quad (c, m, d, m', d') \leftarrow \mathcal{A}(ck);\\
&\quad \_ \ \leftarrow \textit{assert}(m \neq m' \wedge \textit{valid-msg}(m) \wedge \textit{valid-msg}(m'));\\
&\quad b \leftarrow \textit{verify}(vk, m, c, d);\\
&\quad b' \leftarrow \textit{verify}(vk, m', c, d');\\
&\quad \textit{return}(b \wedge b')\} \ \textit{ELSE return}(\textit{False})
\end{aligned} \tag{5.4}$$

Recall that for the scheme to be secure, the Committer adversary should not be able to win this game.

Definition 8. *(Binding Advantage)* $\textit{binding-advantage}(\mathcal{A}) = \mathcal{P}[\textit{binding-game}(\mathcal{A}) = \textit{True}]$

Definition 9. *(Perfect Binding)* $\textit{perfect-binding}(\mathcal{A}) = (\textit{binding-advantage}(\mathcal{A}) = 0)$

5.3 THE RIVEST COMMITMENT SCHEME

In this section we show how we formalise the Rivest commitment scheme [66] introduced by Rivest in 1999. The Rivest scheme is run using a field of prime order, $\mathbb{Z}_q$ and is built using a trusted initialiser. In this case the trusted initialiser provides co-related randomness to the parties in advance of the protocol, it does not participate in the running of the protocol thereafter. Protocols using a trusted initialiser are generally easier to implement as the initialisation can be performed in advance of the protocol and the co-related randomness reduces overheads in the protocol itself.

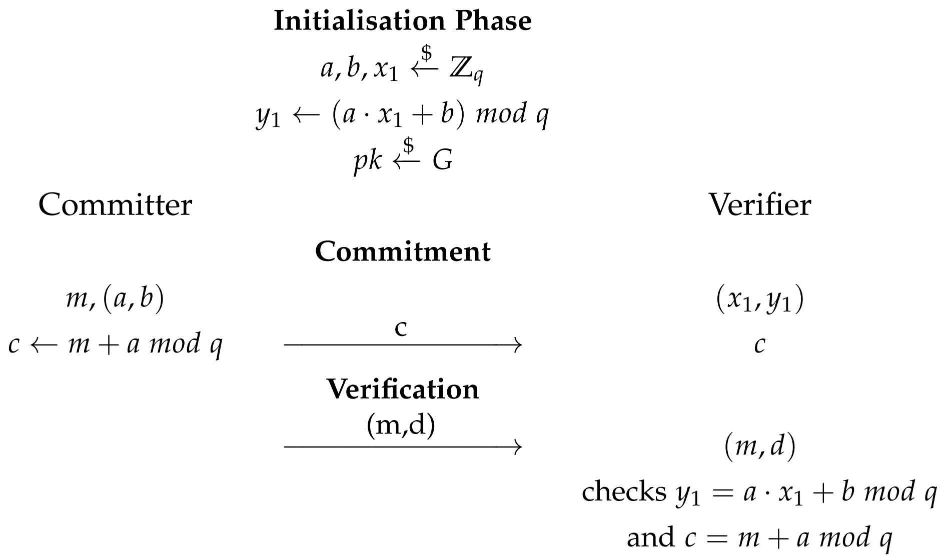

Initialisation Phase
$a, b, x_1 \xleftarrow{\$} \mathbb{Z}_q$
$y_1 \leftarrow (a \cdot x_1 + b) \ mod \ q$
$pk \xleftarrow{\$} G$
Committer
Verifier
Commitment
$m, (a, b)$
(x_1, y_1)
$c \leftarrow m + a \ mod \ q$
c
c
Verification
(m,d)
(m, d)
checks $y_1 = a \cdot x_1 + b \ mod \ q$
and $c = m + a \ mod \ q$

Figure 5.2: The Affine Plane commitment scheme of [17] that slightly amends the Rivest commitment scheme [66].

The protocol we formalise is shown in Figure 5.2. Note this is not quite the original scheme proposed by Rivest [66]; as was noted by Blundo and Masucci in [17] the original scheme did not provide perfect hiding. The original committed message was constructed as $c = a \cdot m + b \ mod \ q$, the latter authors offered a slight amendment that does provide perfect hiding. The trusted initialiser randomly generates a, b and x_1 and constructs $y_1 = a \cdot x_1 + b \ mod \ q$. It sends (a, b) to the Committer and (x_1, y_1) to the Verifier. To commit to the message m the Committer computes $c = m + a \ mod \ q$. To reveal, they send the pair (a, b) and the message m upon which the Verifier checks $c = m + a \ mod \ q$ and $y_1 = a \cdot x_1 + b \ mod \ q$.

We formalise the protocol in the locale *rivest* where we fix the size of the field and assume it is of prime order. Note we do not use any field construction previously formalised in Isabelle, preferring to work modulo q throughout the formalisation.

$$\begin{aligned}&\textbf{locale}\ rivest = \\ &\quad \textbf{fixes}\ q\ ::\ nat \\ &\quad \textbf{assumes}\ prime(q)\end{aligned} \tag{5.5}$$

The components of the commitment scheme are given below. Our formalisation allows for the trusted initialiser as we treat the co-related randomness given to each party as the keys, the work done by the trusted initialiser in the protocol is done in our key generation algorithm.

$$\begin{aligned}&\textit{key-gen}_R = \textit{do}\ \{\\ &\quad a \leftarrow \textit{samp-uniform}(q);\\ &\quad b \leftarrow \textit{samp-uniform}(q);\\ &\quad \textit{let}\ y_1 = (a \cdot x_1 + b)\ \textit{mod}\ q\\ &\quad \textit{return}((a,b),(x_1,y_1))\}\end{aligned} \tag{5.6}$$

The commit and verify algorithms are as defined in Figure 5.2.

$$\textit{commit}_R((a,b),m) = \textit{return}(m + a\ \textit{mod}\ q, (a,b)) \tag{5.7}$$

$$\textit{verify}_R((x_1,y_1),m,c,(a,b)) = (c = m + a\ \textit{mod}\ q \wedge y_1 = a \cdot x_1 + b\ \textit{mod}\ q) \tag{5.8}$$

Finally, a message is considered valid if it is in the valid range of positive naturals — in the field of size q.

$$\textit{valid-msg}_R(m) = m \in \{1,\ldots,q-1\} \tag{5.9}$$

As usual we import the commitment scheme locale, here under the name *rivest-commit*, to show this is an instance with suitable properties.

We first consider the hiding property.

Lemma 17. *(in rivest)* **shows** *rivest-commit.perfect-hiding*$(\mathcal{A})$

Proof. The commitment $c = m + a\ mod\ q$ reveals no information about m as it is masked by the randomness of a, which the Verifier does not have access to. Therefore an application of the one time pad lemma for addition in a field (Equation 5.10), which we prove, means the committed message given to the adversary is independent of the message.

$$\textit{map}(\lambda.\ (c+a)\ \textit{mod}\ q, \textit{samp-uniform}(q)) = \textit{samp-uniform}(q) \tag{5.10}$$

We then show the adversary's guess can be no better than a than flipping a coin to determine its output, meaning its chance of winning the hiding game is $\frac{1}{2}$. □

The binding property is proven by bounding the binding advantage by $\frac{1}{q}$.

Lemma 18. *(in rivest)* **shows** *rivest-commit.bind-advantage*$(\mathcal{A}) \leq \frac{1}{q}$

Proof. The conditions required on the output of the binding adversary (in the binding game) are such that we can compute x_1 (let us call the function computing x_1, f), which is uniformly sampled in the game (as part of the key generation algorithm), from the output of $\mathcal{A}$. Intuitively this means we can correctly guess the output of a uniform sampling from a set of q elements, the probability of which is $\frac{1}{q}$. More formally we have $f(a,a',b,b') = x_1$ where x_1 is a uniform sample. As f is independent of x_1 we show the probability of the game returning true is less than or equal to f guessing the value of x_1, that is the probability is less than $\frac{1}{q}$. □

Correctness comes easily after unfolding the relevant definitions.

Lemma 19. *(in rivest)* **shows** *rivest-commit.correctness*

Together Lemmas 17, 18 and 19 show the desired properties of the commitment scheme presented in Figure 5.2.

The Rivest commitment scheme uses the so called trusted initialiser model. The use of a trusted initialiser to distribute co-related randomness to the parties before the execution of the protocol means protocols can be more simple and still offer high levels of security. As pointed out in [17], however, we still cannot have perfect hiding and binding.

As different commitment schemes have different properties, in particular we can only have either perfect hiding or perfect hiding, we cannot provide an overall definition of commitment schemes in our formalisation. We can only consider the two properties in turn.

6 COMMITMENTS FROM Σ-PROTOCOLS

6.1 INTRODUCTION

As we saw in Section 2.3.1, Σ-protocols can be used to construct commitment schemes. In this chapter we show how we formalise this construction. This general construction means that from any Σ-protocol we get, for free, a commitment scheme. Figure 6.2 outlines the formalisation in this chapter. In particular we use our frameworks for Σ-protocols and commitment schemes to prove the general construction, this is represented by the double arrow between them. Figure 6.2 highlights the instantiation of the Pedersen commitment scheme (which comes using the general construction instantiated for the Schnorr Σ-protocol). In our formalisation we also provide the relevant instantiations for the other Σ-protocols we consider, however here we focus on the Pedersen commitment scheme as it is the most widely used scheme.

CHAPTER OUTLINE We first briefly recap the construction of commitment schemes from Σ-protocols and then show our formalisation of the construction in Section 6.2.1. Then, in Section 6.3.1, we show how instantiate this result for the Pedersen commitment scheme.

The work in this chapter has been published in [22, 24, 25]

6.2 CONSTRUCTING COMMITMENT SCHEMES FROM Σ-PROTOCOLS

Modern cryptography is based on hardness assumptions. These are relations that it is considered computationally infeasible to break. For example, the discrete log assumption (DLP — discrete log problem) given in Equation 2.1.

Consider a hard relation R for a Σ-protocol where gen generates an instance h and witness w such that $R(h, w)$ is satisfied. Using a Σ-protocol for the relation R we can

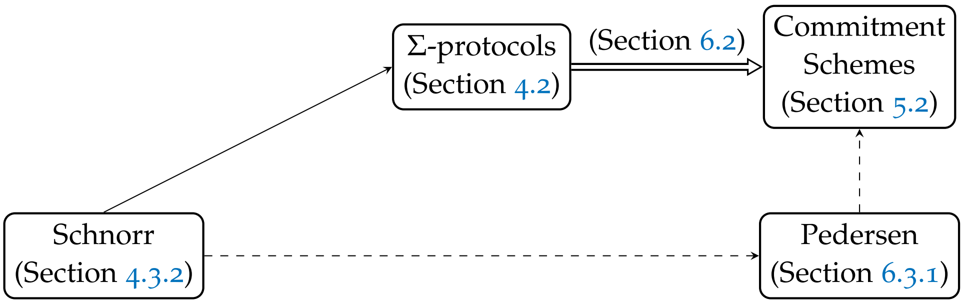


Figure 6.1: Outline of the formalisation of the general proof of the construction of commitment schemes from Σ-protocols, highlighting the instantiation of the Pedersen commitment scheme. We note we also instantiate the other Σ-protocols we consider in the same way.

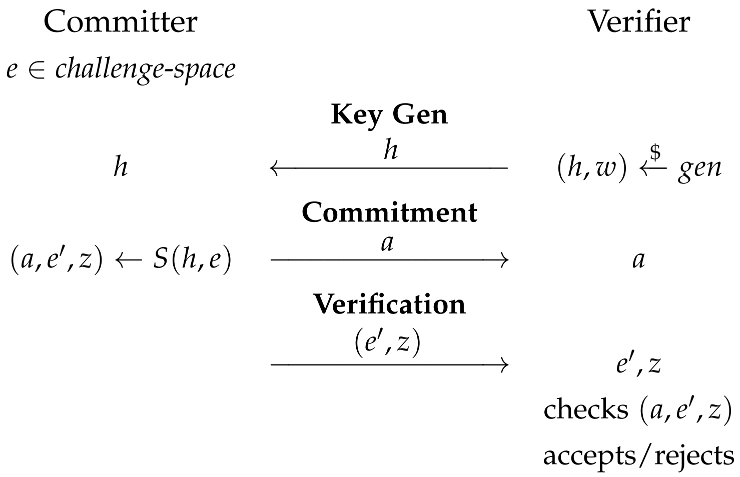


Figure 6.2: A commitment scheme constructed from a Σ-protocol, m is the message being committed to.

construct the commitment scheme given in Figure 6.2. One requirement in that the message to be committed, e, must be in the challenge space of the Σ-protocol. In reality, however, this is not pose a restriction at all as the challenge space is the field or group the Σ-protocol is considered over.

Correctness comes from the HVZK property of the Σ-protocol, the simulator's output is the same as the output of a real execution of the protocol, meaning the check algorithm will accept the conversation. The commitment scheme is perfectly hiding because the commitment a is the first message of the Σ-protocol which is created independently of the challenge (the message being committed to). The binding property follows from the special soundness property of the Σ-protocol; if the Committer could output the commitment a and opening values (e, z) and (e', z') such that both (a, e, z) and (a, e', z') are both accepting conversations then by the special soundness property there exists an adversary that can output the witness w which contradicts the assumption of the relation being hard.

6.2.1 *Formalising the construction*

To formalise this construction we fix the components of a Σ-protocol in a locale and assume they form a Σ-protocol. The locale can be seen in 6.2, where the superscript C denotes we are using the parameters to construct a commitment scheme. The only additional parameter we require in this construction beyond what the Σ-protocol provides is a generator,

$$gen^C \; :: \; (\text{'}pub\text{-}input \times \text{'}witness) \; spmf \tag{6.1}$$

that outputs (h, w) such that the relation is satisfied.

$$\begin{aligned}
&\textbf{locale } \Sigma\textit{-commit} = \Sigma\textit{-protocol-base } init^C\ response^C\ check^C\ Rel^C\ S^C_{raw}\ \mathcal{A}^C_{ss}\\
&\qquad\qquad\qquad\qquad\qquad challenge\text{-}space^C\ valid\text{-}pub^C\\
&\quad \textbf{for } init^C\ response^C\ check^C\ Rel^C\ S^C_{raw}\ \mathcal{A}^C_{ss}\ challenge\text{-}space^C\ valid\text{-}pub^C +\\
&\quad \textbf{and } gen^C\\
&\ \textbf{assumes } \Sigma\text{-}protocol(h, w)\\
&\quad \textbf{and } (h, w) \in set\text{-}spmf(gen^C) \Longrightarrow (h, w) \in Rel^C\\
&\quad \textbf{and } lossless(gen^C)\\
&\quad \textbf{and } lossless(init^C(h, w))\\
&\quad \textbf{and } lossless(response^C(r, w, e))
\end{aligned} \tag{6.2}$$

Using these fixed parameters we make the assumptions they form a Σ-protocol and that the generator outputs a tuple for which the relation holds. The assumptions on the lossessness of the parameters are needed, otherwise the protocol may terminate if they do not output anything; — meaning we cannot reason about the security properties.

To formalise the general notion of a hard relation we define a security game played by an adversary who is trying to break the relation: (h, w) is sampled from gen^C and h is given to the adversary who is asked to output w'. The adversary wins the game if $(h, w') \in Rel^C$.

$$\begin{aligned}
&rel\text{-}game(\mathcal{A}) = TRY\ do\ \{\\
&\quad (h, w) \leftarrow gen^C;\\
&\quad w' \leftarrow \mathcal{A}(h);\\
&\quad return((h, w') \in Rel^C)\}\ ELSE\ return(False)
\end{aligned} \tag{6.3}$$

Using this game we define the relation advantage — the probability an adversary has of winning the game.

Definition 10 (Relation Advantage)**.**

$$rel\text{-}advantage(\mathcal{A}) = \mathcal{P}[rel\text{-}game(\mathcal{A}) = True]$$

We show a reduction to this advantage in the proof of the binding property.

To formalise the protocol given in Figure 6.2 we define the three components $key\text{-}gen^C, commit^C, verify^C$ that make up the commitment scheme and also what constitutes a valid message by defining $valid\text{-}msg^C = (m \in challenge\text{-}space^C)$. The keys are generated by sampling from gen^C.

$$\begin{aligned}
&key\text{-}gen^C = do\ \{\\
&\quad (h, w) \leftarrow G^C;\\
&\quad return(h, (h, w))\}
\end{aligned} \tag{6.4}$$

To commit to a message the Committer runs the simulator and outputs the initial message from the simulator as the commitment and holds the response as the opening value.

$$\begin{aligned} &\mathit{commit}^C(h,e) = \mathit{do}\ \{\\ &\quad (a,e,z) \leftarrow S^C(h,e);\\ &\quad \mathit{return}(a,z)\} \end{aligned} \tag{6.5}$$

Finally the Verifier checks if the messages it has received from the Committer correspond to an accepting conversation.

$$\mathit{verify}^C((h,w),e,a,z) = \mathit{check}^C(h,a,e,z) \tag{6.6}$$

We now prove that our construction of the commitment scheme meets the desired properties. The *commit-base* locale is imported under the name Σ-*commit* thus all definitions are prefixed with this.

sublocale Σ-*commit* : *commit-base key-gen*^{C} *commit*^{C} *verify*^{C} *valid-msg*^{C} .

The formal proofs of the security properties broadly follow the intuition given in Section 6.2. The correctness and hiding properties are given in Lemmas 20 and 21 below.

Lemma 20. *(in* Σ*-commit)* **shows** Σ*-commit.correct*

Proof. We rewrite the simulator that is called in the commitment phase as the real view of the transcript using the HVZK property of Σ-protocols (Definition 2). After unfolding the real view into the components of the Σ-protocol we apply the definition of completeness (Definition 1) to show that check will always return true. □

Lemma 21. *(in* Σ*-commit)* **shows** Σ*-commit.perfect-hiding*$(\mathcal{A})$

Proof. We replace the simulator in the hiding game by the real view of the Σ-protocol. The commitment a comes from the probabilistic program *init*^{C} and is therefore independent of the message that is committed as the only inputs to *init*^{C} are h and w. Thus the adversary learns nothing of the committed message and so the chance of it winning the hiding game is equivalent to guessing the output of a coin flip — which implies perfect hiding. □

Finally we consider the binding property. Here we show a reduction to the relation advantage. To show this reduction we construct an adversary, $\mathit{adversary}_{rel}$, that interacts with the relation game using the Σ-protocols special soundness adversary and the adversary used in the binding game — $\mathit{adversary}_{rel}$ calls the binding adversary and constructs two conversations from it to pass them as inputs to the special soundness adversary and outputs the witness given.

$$\begin{aligned} &\mathit{adversary}_{rel}(\mathcal{A},h) = \mathit{do}\ \{\\ &\quad (c,e,z,e',z') \leftarrow \mathcal{A}(h);\\ &\quad \mathcal{A}^C_{ss}(x,(c,e,z),(c,e',z'))\} \end{aligned} \tag{6.7}$$

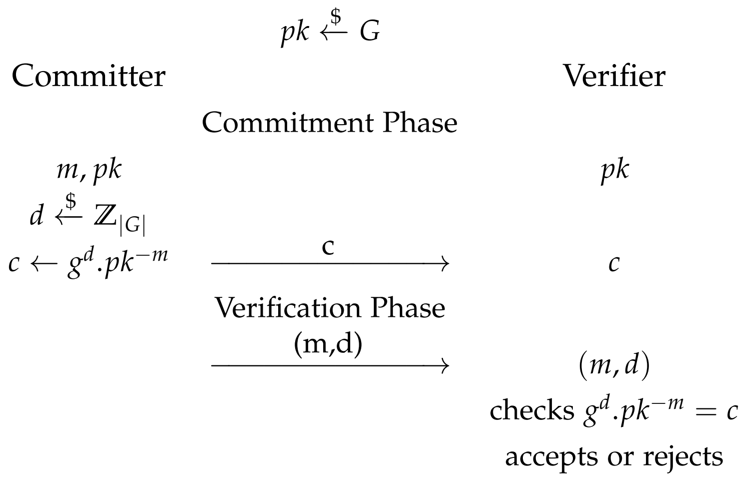


Figure 6.3: The Pedersen commitment protocol, the Committer commits to message m. No keys are known only to one party, we only have a shared key pk.

Lemma 22. *(in Σ-commit)*
shows $\Sigma\textit{-commit.bind-advantage}(\mathcal{A}) \leq \textit{rel-advantage}(\textit{adversary}_{rel}(\mathcal{A}))$

Proof. The binding game is equal to calling $\textit{rel-game}(\textit{adversary}_{rel})$ with the assertions from the binding game incorporated in the probabilistic program. When removing the assertions the probability mass of the probabilistic program can only increase, thus the bound in the above statement is valid. □

The next section details how we use this general proof to realise the commitment schemes constructed from some Σ-protocols — in particular we show how the security statements for the Pedersen commitment scheme come with very little proof effort.

6.3 INSTANTIATING THE GENERAL RESULT

6.3.1 *The Pedersen Commitment Scheme*

The Pedersen commitment scheme is a well known commitment scheme that allows for the commitment to a natural number. In this section we detail how we use our general result from Section 6.2 to realise the proof of the Pedersen commitment scheme in only a few lines of Isabelle proof. The result comes from instantiating the general result with the Schnorr Σ-protocol that was formalised in Section 4.3.2.

We note the exact instantiation of the general result from Section 6.2 outputs a form of the Pedersen scheme that is slightly different from the traditional version presented — given in Figure 2 in Section 2.2. Specifically the commitment is taken as $c = g \cdot pk^{-m}$ rather than $c = g \cdot pk^m$ that is commonly presented in the literature, note the verification step is also modified in the analogous way. This is due to the simulator in the Schnorr protocol taking the inverse of the public input in constructing the initial message. The Pedersen protocol that arises from our formalisation is given in Figure 6.3[1]

To formalise the commitment scheme constructed from the Schnorr Σ-protocol the only additional definition we must make is that of the generator. Recall the generator generates a tuple for which the relation holds. In the case of the Schnorr Σ-protocol the generator G^S is as follows.

$$\begin{aligned} & G^S = do\{ \\ & \quad w \leftarrow sample\text{-}uniform(|G|); \\ & \quad return(g^w, w)\} \end{aligned} \tag{6.8}$$

Here G is the cyclic group we are working with. Along with the instantiated parameters for the Schnorr Σ-protocol we are able to instantiate the locale from the general proof as follows.

$$\textbf{sublocale}\ pedersen : \ \Sigma\text{-}commit\ init^S\ response^S\ check^S$$
$$R^S_{DL}\ S^S_{raw}\ \mathcal{A}^S_{ss}\ challenge\text{-}space^S\ valid\text{-}pub^S\ G$$

To prove this import is valid we must prove that the assumptions from the locale Σ-*commitment* given in Figure 6.2 hold. These properties come easily using Theorem 9 and unfolding the other relevant definitions.

We then directly prove the correctness and hiding statements using the corresponding general lemmas.

Lemma 23. *(in schnorr)*
shows *pedersen.commit-base.correct*

Lemma 24. *(in schnorr)*
shows $pedersen.commit\text{-}base.perfect\text{-}hiding(\mathcal{A})$

Both Lemma 23 and 24 are proven in one line. The binding statement can also be transferred and proved in one line however, we would like to relate it to the discrete log advantage rather than the generic *rel-advantage* given in the general case in Lemma 22. The following lemma shows that the discrete log advantage is equal to the relation advantage defined in the general construction.

Lemma 25. *(in schnorr)*
shows $pedersen.rel\text{-}advantage(\mathcal{A}) = dis\text{-}log.advantage(\mathcal{A})$

Using Lemma 25 we prove binding by bounding the binding advantage by the discrete log advantage.

Lemma 26. *(in schnorr)*
shows $pedersen.commit\text{-}base.bind\text{-}advantage(\mathcal{A}) \leq$
$dis\text{-}log.advantage(pedersen.adversary_{rel}(A))$

[1] There is no obvious notion of equivalence of commitment schemes, but due to the protocols being identical up to a negative exponent we call them both the Pedersen commitment scheme, but note the difference.

```
sublocale pedersen:
  Σ_commit init response check R_DL S2 ss_adversary challenge_space G
  by unfold_locales
    (auto simp add: R_DL_def G_def Schnorr_Σ_inv.L_def sigma_protocol
     lossless_init lossless_response valid_pub_def)

lemma "pedersen.commit_base.correct"
  by(fact pedersen.commit_correct)

lemma "pedersen.commit_base.perfect_hiding_ind_cpa 𝒜"
  by(fact pedersen.perfect_hiding)

lemma rel_adv_eq_dis_log_adv:
  "pedersen.rel_advantage 𝒜 = dis_log.advantage 𝒜"
proof-
  have "pedersen.rel_game 𝒜 = dis_log.dis_log 𝒜"
    unfolding pedersen.rel_game_def R_DL_def dis_log.dis_log_def
    by(auto intro: try_spmf_cong bind_spmf_cong[OF refl]
       simp add: G_def cong_less_modulus_unique_nat group_eq_pow_eq_mod
       finite_carrier pow_generator_eq_iff_cong)
  thus ?thesis
    using pedersen.rel_advantage_def dis_log.advantage_def by simp
qed

lemma bind_advantage_bound_dis_log:
  "pedersen.commit_base.bind_advantage 𝒜 ≤ dis_log.advantage
(pedersen.adversary 𝒜)"
  using pedersen.bind_advantage rel_adv_eq_dis_log_adv by simp
```

Figure 6.4: The Isabelle proof for instantiating the Pedersen commitment scheme using the general proof presented in this section.

6.3.1.1 *The Isabelle Code*

To illustrate how efficient the proof is in Isabelle we show it in its entirety in Figure 6.4, the code has been extracted from Isabelle. For completeness we provide a commentary on the proof here.

First we import, under the name *pedersen*, the locale where the general proof is given and prove the import is valid. The correctness and perfect hiding properties come directly from the general proof, this is seen by the proof that only calls the on the lemmas *pedersen.correct-commit* and *pedersen.perfect-hiding* respectively. For the binding property in the general proof (Lemma 22) we show a reduction to the hard relation, in any instantiation we must relate this to the hardness assumption corresponding to the commitment scheme that has been constructed. In this case we show the relation advantage in the general construction is equivalent to the discrete log advantage. This is shown by the lemma *rel-adv-eq-dis-log-adv*. Using this we can show the binding advantage is bound by the discrete log advantage, thus completing the reduction for the binding property.

6.3.2 *Instantiating the security parameter for the Pedersen Commitment Scheme*

In this section we show how we instantiate the security parameter for the Pedersen commitment scheme. The methodology we use here is analogous to the instantiation of the security parameter in Section 7.3.1.4.

We use Isabelle's locale instantiation mechanism to achieve the instantiation with the security parameters with little effort. First we construct a locale that fixes the family of cyclic groups and then import the *schnorr* locale for all n. That is we construct the locale and make the following assumption and import the concrete setting for all n.

$$\begin{aligned}&\textbf{locale } \textit{schnorr-asymp} =\\ &\quad \textbf{fixes } \mathcal{G} \;::\; \textit{nat} \Rightarrow \textit{'grp cyclic-group } (\textbf{structure})\\ &\quad \textbf{assumes } \textit{schnorr}(G(n))\end{aligned} \tag{6.9}$$

$$\textbf{sublocale } \textit{schnorr}(G(n)) \textbf{ for } n$$

Whilst we are focusing on the Pedersen commitment scheme here, for completeness we give the statement that the Schnorr protocol is a Σ-protocol. The statement in the asymptotic setting comes trivially from the concrete setting.

Theorem 12. *(in schnorr-asymp)* **shows** *Schnorr-Σ.Σ-protocol*(n)

Like usual to prove a commitment scheme secure we consider the three properties in turn. Correctness and perfect hiding for the Pedersen commitment scheme come directly from the concrete setting.

Lemma 27. *(in schnorr-asymp)*
shows *pedersen.commit-base.correct*(n)

Lemma 28. *(in schnorr-asymp)*
shows *pedersen.commit-base.perfect-hiding*$(n, \mathcal{A}(n))$

It is left to show computational binding for the Pedersen commitment scheme. Here we show $\mathcal{A}$'s advantage against the binding game is negligible if *adversary*'s advantage against the discrete log game is negligible. This follows directly from the bound in the concrete case.

Lemma 29. *(in schnorr-asymp)*
shows $\textit{negligible}(\lambda\ n.\ \textit{pedersen.commit-base.bind-advantage}(n, \mathcal{A}(n))) \iff \textit{negligible}(\lambda\ n.\ \textit{dis-log.advantage}(\textit{pedersen.adversary}_{rel}(n, A(n))))$

6.4 OTHER INSTANTIATIONS AND CONCLUSION

OTHER INSTANTIATIONS Section 6.3.1 showed how we instantiated our general result of the construction of commitment schemes from Σ-protocols for the Schnorr Σ-protocol, giving the Pedersen commitment scheme. In this section we show the analogous commitment schemes for the other Σ-protocols we consider; namely the

Chaum-Pedersen and Okamoto Σ-protocols. We note, due to the modularity of the proofs, the results look identical (up to renaming) to those for the Pedersen commitment scheme in Section 6.3.1, thus here we only give the protocols.

In the case of the Chaum-Pedersen Σ-protocol the relation is as in Equation 4.36

$$Rel_{CP} = \{((h_0, h_1), w).\ h_0 = g^w \wedge h_1 = g'^w\}$$

where g and g' are both generators of the cyclic group G. The corresponding commitment scheme is run as follows:

$$\begin{array}{ccc}
\text{Committer} & & \text{Verifier} \\
e \in \{0, \ldots, |G|\} & & \\
 & \textbf{Key Gen} & \\
 & & w \xleftarrow{\$} |G| \\
(g^w, g'^w) & \xleftarrow{\quad (h_0, h_1) \quad} & (h_0, h_1) = (g^w, g'^w) \\
 & \textbf{Commitment} & \\
z \xleftarrow{\$} |G| & & \\
a \leftarrow g^z \cdot h^{-c} & & \\
a' \leftarrow g'^z \cdot h'^{-c} & \xrightarrow{\quad (a, a') \quad} & (a, a') \\
 & \textbf{Verification} & \\
 & \xrightarrow{\quad (e, z) \quad} & e, z \\
 & & \text{checks } a \cdot h_1^e = g^z \wedge a_1 \cdot h_1^e = g'^z \\
 & & \text{accepts/rejects}
\end{array}$$

In the case of the Okamoto Σ-protocol the relation is as in Equation 4.44

$$Rel_{Ok} = \{(h, (w_0, w_1)).\ h = g^{w_0} \wedge h_1 = g^{w_1}\}$$

where g and g' are both generators of the cyclic group G. The corresponding commitment scheme is run as follows:

$$\begin{array}{ccc}
\text{Committer} & & \text{Verifier} \\
e \in \{0,\dots,|G|\} & & \\
 & \textbf{Key Gen} & \\
 & & w_0 \xleftarrow{\$} |G| \\
 & & w_1 \xleftarrow{\$} |G| \\
(g^w, g'^w) & \xleftarrow{(h_0, h_1)} & h = g^{w_0} \cdot g'^{w_1} \\
 & \textbf{Commitment} & \\
z_0 \xleftarrow{\$} |G| & & \\
z_1 \xleftarrow{\$} |G| & & \\
a \leftarrow g^{z_0} \cdot g'^{z_1} \cdot h^{-c} & \xrightarrow{a} & (a, a') \\
 & \textbf{Verification} & \\
 & \xrightarrow{(e,z)} & e, z \\
 & & \text{checks } g^{z_0} \cdot g'^{z_1} = a \cdot h^e \\
 & & \text{accepts/rejects}
\end{array}$$

CONCLUSION This chapter has shown how CryptHOL can be used to prove constructions secure at an abstract level. This methodology is important as it is often at this level that cryptographers work. Our work here has reasoned at a high, general level, assuming only that primitives (Σ-protocols) exist, before instantiating the results, meaning the low level constructions are also formally proven secure. Isabelle allows us to do this with relative ease by leveraging the module system and reusing results where appropriate.

Part II

FORMALISING MULTI-PARTY COMPUTATION

7

SEMI-HONEST SECURITY

7.1 INTRODUCTION

The semi-honest adversary model assumes the adversary, who corrupts a party in a protocol, follows the protocol transcript but may try to learn more than is allowed by analysing the messages it receives. This security model is a baseline for security in MPC, in particular protocols that have semi-honest security can be extended to have malicious security where the adversary is allowed to totally corrupt the party. This Chapter details our work on formalising MPC in the semi-honest model — throughout we consider the two party setting. We refer the reader back to Section 2.4.1 for a detailed account of the definitions.

CHAPTER OUTLINE In Section 7.2 we show how we formalise the definitions of semi-honest security from section 2.4.1. In Section 7.3 we consider the Oblivious Transfer functionality and prove instantiations of it: namely the OT_2^1 constructed from Extended Trapdoor Permutations (ETPs) and the Naor Pinkas OT_2^1. Section 7.4 shows how we prove OT_4^1 from OT_2^1 and subsequently use this to prove the two party GMW protocol secure. Finally in Section 7.5 we consider a protocol for securely computing a multiplication between two parties. The contributions we make are listed below and a visual outline of the Chapter is given in Figure 7.1.

The work in this chapter has been published in [20, 21, 23].

7.2 FORMALISING SEMI-HONEST SECURITY

In this section we present our formalisation of the definitions of semi-honest security.

CryptHOL cannot reason about the complexity class of adversaries, in particular we cannot reason about polynomial-runtime. As discussed in section 3.3.2 this does not effect the structure of our proofs as we can still employ a reduction based argument. Without this, however, we do not consider computational indistinguiability as described in section 2.4.1. Instead we opt for the standard (among other cryptographic frameworks) notion of an *advantage* — the advantage a distinguisher has of distinguishing the two views. In CryptHOL the notion of an advantage was introduced by Lochbihler et al. [8, 53] when they formalised game-based cryptography.

We consider the case of deterministic and non-deterministic functionalities separately.

7.2.1 *Deterministic functionalities*

Recall that for deterministic functionalities the notion of security is slightly relaxed, however correctness must be proved separately. The locale which fixes the parameters for this case is given in 7.1.

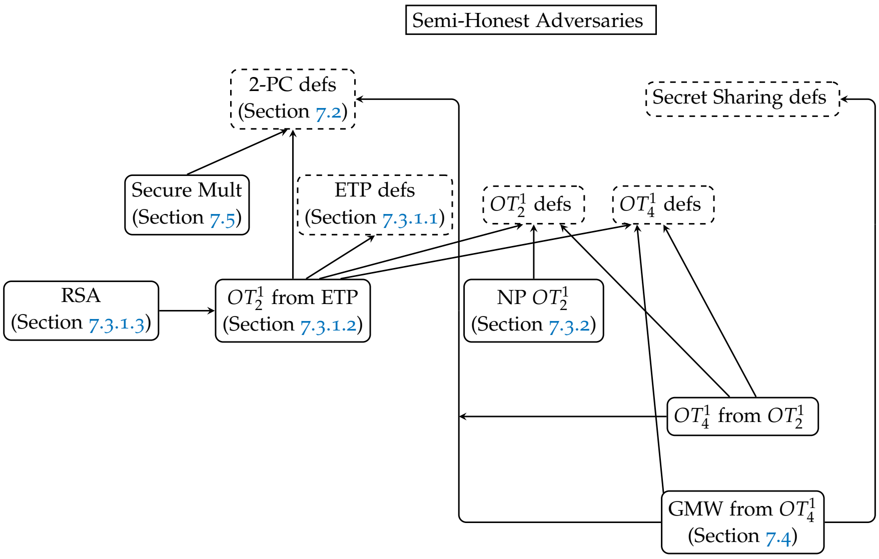

Semi-Honest Adversaries
2-PC defs (Section 7.2)
Secret Sharing defs
Secure Mult (Section 7.5)
ETP defs (Section 7.3.1.1)
OT^1_2 defs
OT^1_4 defs
RSA (Section 7.3.1.3)
OT^1_2 from ETP (Section 7.3.1.2)
NP OT^1_2 (Section 7.3.2)
OT^1_4 from OT^1_2
GMW from OT^1_4 (Section 7.4)

Figure 7.1: An outline of the formalisation of semi-honest MPC in this thesis.

$$\begin{aligned}
&\textbf{locale } \textit{semi-honest-det} = \\
&\quad \textbf{fixes } \textit{funct} :: \textit{'msg}_1 \Rightarrow \textit{'msg}_2 \Rightarrow (\textit{'out}_1 \times \textit{'out}_2)\ \textit{spmf} \\
&\qquad \textbf{and } \textit{protocol} :: \textit{'msg}_1 \Rightarrow \textit{'msg}_2 \Rightarrow (\textit{'out}_1 \times \textit{'out}_2)\ \textit{spmf} \\
&\qquad \textbf{and } R_1 :: \textit{'msg}_1 \Rightarrow \textit{'msg}_2 \Rightarrow \textit{'view}_1\ \textit{spmf} \\
&\qquad \textbf{and } S_1 :: \textit{'msg}_1 \Rightarrow \textit{'out}_1 \Rightarrow \textit{'view}_1\ \textit{spmf} \\
&\qquad \textbf{and } R_2 :: \textit{'msg}_1 \Rightarrow \textit{'msg}_2 \Rightarrow \textit{'view}_2\ \textit{spmf} \\
&\qquad \textbf{and } S_2 :: \textit{'msg}_2 \Rightarrow \textit{'out}_2 \Rightarrow \textit{'view}_2\ \textit{spmf}
\end{aligned} \tag{7.1}$$

The parameter *funct* represents the functionality, taking in the party's inputs and outputting the respective outputs.

The parameter, *protocol*, represents the execution of the protocol modelled as a probabilistic program which outputs the outputs of the respective parties, as constructed in the protocol. It is this parameter we compare to the functionality to define correctness. The views are captured by the parameters R_1, S_1, R_2, S_2. The definitions of security are made with respect to these parameters.

To define security in the deterministic setting we first consider correctness.

CORRECTNESS A protocol is correct if it is functionally equivalent to the functionality it implements.

Definition 11 (Correctness)**.**

$$correct(m_1, m_2) = (protocol(m_1, m_2) = funct(m_1, m_2))$$

Here m_1 and m_2 are the inputs to the protocol for party 1 and 2. Proofs of correctness are generally proven by unfolding the relevant definitions and providing Isabelle with some hints on how to rewrite some terms. Depending on the protocol, Isabelle requires more or less help with the rewriting steps; more help is needed when the steps require non-trivial assumptions. Looking at the proof scripts will show how sometimes Isabelle really does need some very basic help to rewrite terms!

SECURITY To formally prove security we construct the ideal view for each party. In the semi-honest setting this is just the output of the simulator for each party. Recall the simulator is allowed two inputs, the input of the party it is simulating, and the output of the functionality of that party. Thus to define the ideal view we sample from the functionality and use the binding operator to hand the relevant output to the simulator as seen in Equation 7.2. The binding notation is recapped below.

$$ideal_1(m_1, m_2) = funct(m_1, m_2) \triangleright (\lambda(out_1, out_2).\ S_1(m_1, out_1)). \tag{7.2}$$

The right hand side of the statement can be read as: the output distribution of the simulator (S_1) on input m_1 and the output for party 1 (out_1) that has been sampled from the functionality. More explicitly, using the monadic do notation this reads:

$$do\ \{(out_1, out_2) \leftarrow funct(m_1, m_2); S_1(m_1, out_1)\}.$$

The analogous definition is made for party 2 also:

$$ideal_2(m_1, m_2) = funct(m_1, m_2) \triangleright (\lambda(out_1, out_2).\ S_2(m_2, out_2)). \tag{7.3}$$

Without looking closely at the definition it may seem suspicious that the ideal view takes in the inputs of both parties — after all the idea is for the ideal world not to take in the input of the other party. This input, however, is not given to the simulator, only to the functionality.

Perfect security requires the real and simulated views to be equal. We define this for party 1 below:

Definition 12 (Perfect security, for party 1).

$$perfect\text{-}sec\text{-}P_1(m_1, m_2) \equiv (R_1(m_1, m_2) = ideal_1(m_1, m_2))$$

We make the analogous definition for party 2.

Definition 13 (Perfect security, for party 2).

$$perfect\text{-}sec\text{-}P_2(m_1, m_2) \equiv (R_2(m_1, m_2) = ideal_2(m_1, m_2))$$

When perfect security cannot be proven we instead show that the probability of a distinguisher (in principle in polytime) distinguishing the real and simulated views — we call this probability the *advantage*. We define the advantage of a distinguisher, D, for party 1 as follows.

Definition 14 (Advantage for party 1).

$$adv\text{-}P_1(m_1, m_2, D) = (|\mathcal{P}[(R_1(m_1, m_2) \triangleright D) = True] - \mathcal{P}[(ideal_1(m_1, m_2) \triangleright D) = True]|)$$

We also provide the analogous definition for party 2.

Definition 15 (Advantage for party 2)**.**

$$\textit{adv-P}_2(m_1, m_2, D) = (|\mathcal{P}[(R_2(m_1, m_2) \rhd D) = \textit{True}] - \mathcal{P}[(\textit{ideal}_2(m_1, m_2) \rhd D) = \textit{True}]|)$$

Clearly, if we have perfect security then the advantage is zero. When considering security with respect the advantages we need to show they are small. To do this we prove a reduction to a known hard problem. When we instantiate our proofs in the asymptotic setting we show the advantage is negligible.

7.2.2 *Non-Deterministic Functionalities*

When considering non-deterministic functionalities the definitions of security are tightened. Specifically the views must also include the output of the functionality and protocol respectively. In this section we show how we incorporate this into our framework. While the framework we provide for this case looks similar to the deterministic setting it is not clear we can reuse the work as the underlying structure of the parameters is different. For example, the output type of R_1 is now appended with the output of the protocol. The locale *semi-honest-non-det* in 7.4 shows how we we fix the required parameters in this setting.

$$\begin{aligned}
&\textbf{locale}\ \textit{semi-honest-non-det} = \\
&\quad \textbf{fixes}\ \textit{funct} :: \textit{'msg}_1 \Rightarrow \textit{'msg}_2 \Rightarrow (\textit{'out}_1 \times \textit{'out}_2)\ \textit{spmf} \\
&\qquad \textbf{and}\ R_1 :: \textit{'msg}_1 \Rightarrow \textit{'msg}_2 \Rightarrow (\textit{'view}_1 \times (\textit{'out}_1 \times \textit{'out}_2))\ \textit{spmf} \\
&\qquad \textbf{and}\ S_1 :: \textit{'msg}_1 \Rightarrow \textit{'out}_1 \Rightarrow \textit{'view}_1\ \textit{spmf} \\
&\qquad \textbf{and}\ \textit{Out}_1 :: \textit{'msg}_1 \Rightarrow \textit{'msg}_2 \Rightarrow \textit{'out}_1 \Rightarrow (\textit{'out}_1 \times \textit{'out}_2)\ \textit{spmf} \\
&\qquad \textbf{and}\ R_2 :: \textit{'msg}_1 \Rightarrow \textit{'msg}_2 \Rightarrow (\textit{'view}_2 \times (\textit{'out}_1 \times \textit{'out}_2))\ \textit{spmf} \\
&\qquad \textbf{and}\ S_2 :: \textit{'msg}_2 \Rightarrow \textit{'out}_2 \Rightarrow \textit{'view}_2\ \textit{spmf} \\
&\qquad \textbf{and}\ \textit{Out}_2 :: \textit{'msg}_2 \Rightarrow \textit{'msg}_1 \Rightarrow \textit{'out}_2 \Rightarrow (\textit{'out}_1 \times \textit{'out}_2)\ \textit{spmf}
\end{aligned} \tag{7.4}$$

The major difference to the deterministic case are the types of the views. The real views take both inputs and output the view of the party along with the output of the protocol. When constructing the ideal view we had a design choice to make. We could fix the type of the ideal view and allow the user to instantiate it — that is we could fix

$$\textit{ideal}_i :: \textit{'msg}_1 \Rightarrow \textit{'msg}_2 \Rightarrow (\textit{'view}_i \times (\textit{'out}_1 \times \textit{'out}_2))\ \textit{spmf}. \tag{7.5}$$

However we choose to fix as parameters the simulators (S_1, S_2) and the *output* functions $(\textit{Out}_1, \textit{Out}_2)$ that construct the output of the protocol in the manner of the functionality. We make this design choice so the simulator can be defined by the user, in an instantiation, without the risk of using an illegal input; namely the input from the other party. If we fixed the ideal view (in the locale) as suggested in Equation 7.5 the user could *accidentally* construct a simulator that depends on the input from the other party, which of course is not allowed. In this way we safeguard the integrity of our framework against *malicious* users[1] and erroneous proofs.

[1] While this would pose no issue if the user of the formalisation was *honest*, given the area of research this work is considering we probably have a moral duty to assume the user is not *honest*.

Using our fixed parameters (given in the locale in 7.4) we construct the ideal view for both parties. We note they indeed have the same type as given in Equation 7.5 but the simulator, which is sampled from inside the ideal view, takes only the allowed inputs.

$$\begin{aligned} &\textit{ideal-view}_1(m_1, m_2, out_1) \equiv do\ \{ \\ &\quad view_1 \leftarrow S_1(m_1, out_1) \\ &\quad out \leftarrow Out_1(m_1, m_2, out_1); \\ &\quad return(view_1, out)\} \end{aligned} \tag{7.6}$$

$$\begin{aligned} &\textit{ideal-view}_2(m_1, m_2, out_2) \equiv do\ \{ \\ &\quad view_2 \leftarrow S_2(m_2, out_2) \\ &\quad out \leftarrow Out_2(m_2, m_1, out_2); \\ &\quad return(view_2, out)\} \end{aligned} \tag{7.7}$$

Now the ideal view has been defined we make the definitions of security. These are the same as for the deterministic setting. The types of the views are now different and we do not need to consider the property of correctness; this is because the output of the protocol is now incorporated into the views. As in the deterministic case we define the ideal world for each party. Here the output of the functionality is input to the ideal view (in the deterministic case this the simulator).

$$ideal_1(m_1, m_2) = funct(m_1, m_2) \triangleright (\lambda(out_1, out_2).\ \textit{ideal-view}_1(m_1, m_2, out_1)). \tag{7.8}$$

$$ideal_2(m_1, m_2) = funct(m_1, m_2) \triangleright (\lambda(out_1, out_2).\ \textit{ideal-view}_2(m_2, m_1, out_2)). \tag{7.9}$$

Definition 16 (Perfect security, for party 1).

$$\textit{perfect-sec-P}_1(m_1, m_2) = (R_1(m_1, m_2) = ideal_1(m_1, m_2))$$

We make the analogous definition for party 2.

Definition 17 (Perfect security, for party 2).

$$\textit{perfect-sec-P}_2(m_1, m_2) = (R_2(m_1, m_2) = ideal_2(m_1, m_2))$$

Analogous to the deterministic case we define the advantage for each party also.

Definition 18 (Advantage for party 1).

$$\textit{adv-P}_1(m_1, m_2, D) = (|\mathcal{P}[(R_1(m_1, m_2) \triangleright D) = True] - \mathcal{P}[(ideal_1(m_1, m_2) \triangleright D) = True]|)$$

Definition 19 (Advantage for party 2).

$$\textit{adv-P}_2(m_1, m_2, D) = (|\mathcal{P}[(R_2(m_1, m_2) \triangleright D) = True] - \mathcal{P}[(ideal_2(m_1, m_2) \triangleright D) = True]|)$$

Our frameworks (the deterministic and non-deterministic case) require the user to input the definitions of the parameters for any instantiation they wish to prove, that is the parameters for the respective locale being used. Therefore the *human verifier* must check that this has been done correctly and accurately.

7.2.3 *Equivalence to EasyCrypt Definitions*

Almeida et al. [1] define semi-honest security using a game where a bit is flipped to determine which view the distinguisher is given. In this section we show our definitions are equivalent to theirs. We transcribe the definitions from the EasyCrypt definitions into Isabelle and prove the equivalence. While there is no guarantee that the EasyCrypt definitions are transcribed correctly (other than observation) the proof of equivalence adds confidence to both sets of definitions. We prove the equivalence for the deterministic case, the proof would be analogous for the non-deterministic setting.

We transcribe the EasyCrypt definitions as follows, noting R_1, S_1, R_2 and S_2 are the same as in our definitions.[2]

Definition 20 (EasyCrypt definition for party 1 (transcribed to Isabelle))**.** *To define the advantage given in EasyCrypt, $adv_{EC,P1}$, we first define the security game for party 1.*

$$\begin{aligned}
&game_{P1}(m_1, m_2, D) \equiv do\ \{\\
&\quad b \leftarrow coin;\\
&\quad (out_1, out_2) \leftarrow funct(m_1, m_2);\\
&\quad r_{view} \leftarrow R_1(m_1, m_2);\\
&\quad s_{view} \leftarrow S_1(m_1, out_1);\\
&\quad b' \leftarrow D(if\ b\ then\ r_{view}\ else\ s_{view});\\
&\quad return(b = b')\}
\end{aligned}$$

$$adv_{EC,P1}(m_1, m_2, D) = |2 \cdot \mathcal{P}[game_{P1}(m_1, m_2, D) = True] - 1|$$

Definition 21 (EasyCrypt definition for party 2 transcribed to Isabelle)**.** *To define the advantage given in EasyCrypt, $adv_{EC,P2}$, we first define the security game for party 2.*

$$\begin{aligned}
&game_{P2}(m_1, m_2, D) \equiv do\ \{\\
&\quad b \leftarrow coin;\\
&\quad (out_1, out_2) \leftarrow funct(m_1, m_2);\\
&\quad r_{view} \leftarrow R_2(m_1, m_2);\\
&\quad r_{view} \leftarrow S_2(m_1, out_1);\\
&\quad b' \leftarrow D(if\ b\ then\ r_{view}\ else\ s_{view});\\
&\quad return(b = b')\}
\end{aligned}$$

$$adv_{EC,P2}(m_1, m_2, D) = |2 \cdot \mathcal{P}[game_{P2}(m_1, m_2, D) = True] - 1|$$

We show these definitions are equivalent to the definitions in our deterministic framework. To show this we must assume the distinguishers and the views are lossless.

Lemma 30. *(in semi-honest-det)*
assumes $\forall view.\ lossless\text{-}spmf(D(view))$

[2] We make the definitions inside the locale given in Figure 7.1 (*semi-honest-det*).

and $\forall m_1\ m_2.\ \mathit{lossless\text{-}spmf}(R_1(m_1, m_2))$
and $\forall m_1\ out_1.\ \mathit{lossless\text{-}spmf}(S_1(m_1, out_1))$
shows $adv_{EC,P1}(m_1, m_2, D) = adv\text{-}P_1(m_1, m_2, D)$

Proof. The proof requires us to split the probability based on the coin flip in the security game given in Definition 20. Intuitively this is where the multiplication by 2 comes from in the advantage in Definition 20. Once we have made the case split on the coin flip in one case the distinguisher gets the real view and in the other it gets the simulated view, which is the same as the definition of the advantage we provide. We note the proof was shortened[3] by the use of Isabelle's monad normalisation theory [67] which provides automation for the reordering of samples. □

Lemma 31. *(in semi-honest-det)*
assumes $\forall view.\ \mathit{lossless\text{-}spmf}(D(view))$
and $\forall m_1\ m_2.\ \mathit{lossless\text{-}spmf}(R_2(m_1, m_2))$
and $\forall m_2\ out_2.\ \mathit{lossless\text{-}spmf}(S_2(m_2, out_2))$
shows $adv_{EC,P2}(m_1, m_2, D) = adv\text{-}P_2(m_1, m_2, D)$

Proof. The proof is analogous to the proof of Lemma 30. □

In this section we have defined our frameworks for reasoning about semi-honest security in the two party setting. In the rest of this Chapter we will see how we use these to realise the security of well known protocols. We first consider the fundamental building block of MPC, Oblivious Transfer.

7.3 1-OUT-OF-2 OBLIVIOUS TRANSFER

In this section we prove two OT^1_2 protocols secure with respect the definitions we give in section 7.2. First we consider a general OT^1_2 protocol constructed from Extended Trapdoor Permutations (ETPs) [35], this is the first protocol Lindell considers in his tutorial [48]. We instantiate our general proof for a known ETP, the RSA function, in section 7.3.1.3. Our general proof means the RSA instantiation only requires us to prove the RSA construction is an ETP. Second we consider the Naor-Pinkas OT^1_2 [57], which is based on the DDH assumption: we prove it secure in section 7.3.2.

Oblivious Transfer (OT) was first introduced by Rabin in [65] and is a fundamental cryptographic primitive that is central to many MPC protocols. It is a two party protocol run between a Sender and a Receiver. In the general case the Sender holds n many messages and the Receiver a choice of k messages. The output of the protocol is that the Receiver obtains their choice of k messages and the Sender obtains nothing.

OT comes in many flavours, the most general being *k-out-of-n OT* (OT^k_n, which we have just described). We examine OT^1_2 as it is the most fundamental form, and can be used to construct other forms of OT — we will see this in section 7.4 where we use OT^1_2 to construct OT^1_4 in the GMW protocol.

Recall, from Example 3 the functionality for OT^1_2 is given by:

$$f_{OT^1_2}((m_0, m_1), \sigma) = (\_, m_\sigma). \tag{7.10}$$

[3] By shortened we not only mean shortened in length, but also time. Reordering samples within a probabilistic program is time consuming and intricate (and very boring!).

This is encoded into Isabelle as

$$funct_{OT^1_2}((m_0, m_1), \sigma) = return(\_, if\ \sigma\ then\ m_1\ else\ m_0). \tag{7.11}$$

Recall that we require our functionalities (even the deterministic ones) to return an *spmf*.

7.3.1 *ETP based OT^1_2*

We first introduce ETPs and how we define their corresponding security properties. We provide informal definitions before presenting the formalisation.[4]

7.3.1.1 *ETPs and HCPs*

An Extended Trapdoor Permutation (ETP) is a collection of permutations $\{f_\alpha\}_\alpha$ along with four algorithms I (index), S (sample), F (forward) and F^{-1} (backward). Here we follow [38, Appendix C.1] in introducing them[5], but note that there is also a brief description of them in [49, Section 4.3].

Definition (informal) 14. *The algorithms that comprise an Extended Trapdoor Permutation (ETP) are as follows:*

- *$I(n)$ samples an index α of a permutation, f_α, as well as a corresponding trapdoor τ for the permutation, $(\alpha, \tau) \leftarrow I(n)$. Here n is the security parameter: in our formalisation this is assumed to be implicit.*
- *$S(\alpha)$ samples an (almost) uniform element in the domain of f_α, in the literature $S(\alpha; r)$ denotes the output of $S(\alpha)$ with random tape r.*
- *F performs the mapping of f_α, $F(\alpha, x) = f_\alpha(x)$ when x is in the domain of f_α and (α, τ) is an output of I.*
- *F^{-1} computes the inverse of f_α, $F^{-1}(\alpha, \tau, y) = f_\alpha^{-1}(y)$ for y in the range of f_α and $(\alpha, \tau) \leftarrow I(n)$.*

These functions can be thought of as a collection of one way permutations with a trapdoor with which the inverse can be obtained easily — and which without the trapdoor, the inverse is computationally infeasible to obtain.

The definition of S in the literature requires the output to be *almost* uniform in the domain (which equals the range as f is a permutation) of f_α. The notion of almost uniform is not defined in either [38] or [49]. This property of S is only used for the security of party 1; namely, the almost uniform nature of the sample means there is statistical closeness between the views. In our formalisation we take S to sample uniformly and can therefore show perfect security for party 1. In the instantiation

[4] Throughout the rest of the thesis we do not explicitly state and explain the sublocale structures in our work. Like Part i we still give all the major locales we use, however do not give every sublocale where we feel the context is clear and adds nothing that has not already been learned from Part i. Therefore we expense with the '.' notation.

[5] Goldreich [38] uses B for the backward algorithm, whereas we use it to denote the hard-core predicate, in this sense our notation is in line with Lindell's [48].

of RSA as an ETP S, is a uniform sample from the field of order N where N is the RSA modulus and therefore we argue that defining S to be a uniform sampling is reasonable in this case.

It should also be noted that the definition of S provided in [38] and [49] uses values of randomness as inputs meaning S is considered to be deterministic. However, there is no need for such input in our formalisation as we model S (and I) as probabilistic programs that toss their own random coins.

The example of ETPs we instantiate is the RSA construction, we introduce this informally in the example below.

Example 5 (RSA ETP)**.** *The RSA function provides an example of an ETP.*

The RSA function is considered on input (N, e):

$$F_{RSA}((N,e),x) = x^e \ mod \ N \tag{7.12}$$

where $N = P \cdot Q$ for primes P and Q and e is such that $gcd(e,(P-1)\cdot(Q-1)) = 1$. That is, I_{RSA} must output (N,e) as the index. The trapdoor τ is the multiplicative inverse of e mod $(P-1)\cdot(Q-1)$. F_{RSA}^{-1} is constructed as follows:

$$F_{RSA}^{-1}((N,e),d,y) = y^d \ mod \ N. \tag{7.13}$$

The following property must hold for any trapdoor function and its inverse. The property is of course obvious if the inverse is used but it is not always so trivial when the trapdoor inverse function is used.

$$F_{RSA}^{-1}((N,e),d,F_{RSA}((N,e),x)) = x \tag{7.14}$$

We provide a formal proof of this property in section 7.3.1.3. The range and domain of the RSA ETP are the same (it is a permutation), namely $\{0,\dots N-1\}$ and $S(N,e)$ outputs a uniform sample from this set.

In reality, when implementing RSA there are certain parameters that need to be avoided (e.g. see [60]). The issue concerns the selection of the prime numbers P and Q. In our instantiated formalisation here we parameterise over a *prime-set* from which we sample. Thus, at an abstract level, we can assume that this set is void of any *weak* parameters.

Associated with an ETP is a Hard Core Predicate (HCP), B. Intuitively, B is an HCP of f if, given $f(\alpha,x)$ for a uniformly sampled x, an adversary cannot distinguish $B(\alpha,x)$ from a random bit. We take the definition from [49].

Definition (informal) 15. *[Hard-Core Predicate] B is a hard-core predicate (HCP) of (I,S,F,F^{-1}) if for every probabilistic polynomial time $\mathcal{A}$ there exists a negligible function μ such that for every n we have,*

$$Pr[\mathcal{A}(n,\alpha,r) = B(\alpha,f^{-1}(S(\alpha;r)))] \leq \frac{1}{2} + \mu(n)$$

where $(\alpha,\tau) \leftarrow I(n)$.

FORMALISING ETPS Our formalisation of ETPs fixes five parameters in the locale *etp-base*: I, *domain*, *range*, F, F^{-1} and B.

$$
\begin{aligned}
&\textbf{locale } \textit{etp-base} = \\
&\quad \textbf{fixes } I :: (\textit{'index} \times \textit{'trap})\ \textit{spmf} \\
&\qquad \textbf{and } \textit{domain} :: \textit{'index} \Rightarrow \textit{'domain set} \\
&\qquad \textbf{and } \textit{range} :: \textit{'index} \Rightarrow \textit{'range set} \\
&\qquad \textbf{and } F :: \textit{'index} \Rightarrow (\textit{'domain} \Rightarrow \textit{'range}) \\
&\qquad \textbf{and } F^{-1} :: \textit{'index} \Rightarrow \textit{'trap} \Rightarrow (\textit{'range} \Rightarrow \textit{'domain}) \\
&\qquad \textbf{and } B :: \textit{'index} \Rightarrow \textit{'range} \Rightarrow \textit{bool} \\
&\quad \textbf{assumes } (\alpha, \tau) \in \textit{set-spmf}(I) \longrightarrow \textit{domain}(\alpha) = \textit{range}(\alpha) \\
&\qquad \textbf{and } (\alpha, \tau) \in \textit{set-spmf}(I) \longrightarrow \textit{finite}(\textit{range}(\alpha)) \\
&\qquad \textbf{and } (\alpha, \tau) \in \textit{set-spmf}(I) \longrightarrow \textit{range}(\alpha) \neq \{\} \\
&\qquad \textbf{and } (\alpha, \tau) \in \textit{set-spmf}(I) \longrightarrow \textit{bij-betw}(F(\alpha), \textit{domain}(\alpha), \textit{range}(\alpha)) \\
&\qquad \textbf{and } (\alpha, \tau) \in \textit{set-spmf}(I) \longrightarrow x \in \textit{range}(\alpha) \longrightarrow F^{-1}(\alpha, \tau, (F(\alpha, x))) = x \\
&\qquad \textbf{and } \textit{lossless-spmf}(I)
\end{aligned}
\tag{7.15}
$$

The assumptions made in *etp-base* capture the properties required of ETPs from Definition 14 as well as the property of the inverse given in Example 5 in Equation 7.14 (here the property is instantiated for the RSA construction). There is no need to make any assumption on B (other than it exists) as we define the security property we require of it later and then use this to show the reduction in the general proof in section 7.3.1.2.

We define S below.

Definition 22. $S(\alpha) = \textit{uniform}(\textit{range}(\alpha))$

We prove an important property on S and F, namely that F applied to a sample from S produces the same distribution as S itself. While this property is trivial to reason about on paper, as F is a permutation, the formal reasoning is more in depth.

Lemma 32. *(in etp-base)*

$$
\begin{aligned}
&\textbf{assumes } (\alpha, \tau) \in \textit{set-spmf}(I) \\
&\textbf{shows } \textit{map-spmf}((\lambda x.\ F(\alpha, x)), S(\alpha)) = S(\alpha)
\end{aligned}
$$

Proof. The left hand side is the same as the uniform sample from the set formed by the image of F on the range. This set is, in turn, the same as the range as F is a permutation. Using both of these we show the left hand side is equal to taking a uniform sample from the range, which, after unfolding the definition of S is the same as the left hand side. □

To formalise the security property of HCPs given in Definition 15 we make the definition of the HCP advantage adv_{HCP}. This captures the probability that $\mathcal{A}$ wins the HCP game. The aim of the adversary $\mathcal{A}$ in the HCP game is to guess the value of B — the HCP fixed in the locale *etp-base* (Equation 7.15) — and thus to beat the HCP assumption. The game is defined as follows:

$$\begin{aligned}
&HCP_{game}(\mathcal{A}, \sigma, b_\sigma, D) \equiv do \ \{ \\
&\quad (\alpha, \tau) \leftarrow I; \\
&\quad x \leftarrow S(\alpha); \\
&\quad let \ b = B(\alpha, F^{-1}(\alpha, \tau, x)); \\
&\quad b' \leftarrow \mathcal{A}(\alpha, \sigma, b_\sigma, x, D); \\
&\quad return(b = b')\}
\end{aligned} \tag{7.16}$$

In the HCP game $\mathcal{A}$ receives α, σ and b_σ. as input In addition we must pass x to $\mathcal{A}$, this is because we do not carry around the randomness given to S. The HCP advantage is defined below.

Definition 23 (HCP advantage)**.**

$$adv_{HCP}(\mathcal{A}, \sigma, b_\sigma, D) = |\mathcal{P}[HCP_{game}(\mathcal{A}, \sigma, b_\sigma, D) = True] - \frac{1}{2}|$$

7.3.1.2 *Realising OT_2^1 using ETPs*

The OT_2^1 protocol we consider is described in Protocol 1 below.

Protocol 1. *P_1 has input $(b_0, b_1) \in \{0,1\}$, P_2 has input $\sigma \in \{0,1\}$ and n is the security parameter.*

1. *P_1 samples an index and trapdoor, $(\alpha, \tau) \leftarrow I(n)$, and sends the index, α, to P_2.*
2. *P_2 samples S twice, $x_\sigma \leftarrow S(\alpha)$, $y_{1-\sigma} \leftarrow S(\alpha)$ and sets $y_\sigma = F(\alpha, x_\sigma)$.*
3. *P_2 sends y_0 and y_1 to P_1.*
4. *P_1 computes $x_0 = F^{-1}(\alpha, \tau, y_0)$, $x_1 = F^{-1}(\alpha, \tau, y_1)$, $\beta_0 = B(\alpha, x_0) \oplus b_0$ and $\beta_1 = B(\alpha, x_1) \oplus b_1$.*
5. *P_1 sends β_0, β_1 to P_2.*
6. *P_2 computes $b_\sigma = B(\alpha, x_\sigma) \oplus \beta_\sigma$*

Intuitively, party 2 samples $y_\sigma, y_{1-\sigma}$ where it only knows the pre-image of one. party 1 then inverts both pre-images (as it knows the trapdoor) and sends both its input messages to party 2 masked by the HCP of the inverted pre-images. party 2 can obtain its chosen message as it knows the inverse of the pre-image but learns nothing of the other message as it cannot guess the HCP (with probability greater than $\frac{1}{2}$). party 1 learns nothing of party 2's choice bit as it only receives $y_\sigma, y_{1-\sigma}$ which share an equal distribution.

To formalise Protocol 1 we construct a locale that uses the *etp-base* locale as follows.

$$\textbf{locale} \ \textit{etp-ot} = \textit{etp} : \ \textit{etp-base I domain range F F}^{-1} \ B \tag{7.17}$$

In this locale the assumptions made on the parameters of the *etp-base* locale are available, so we know the parameters form an ETP.

We formalise the execution of the protocol with the following probabilistic program. Note the security parameter does not appear as we instantiate it (as an input to I) later.

$$\begin{aligned}
&protocol_{OT^1_2,ETP}\ ((b_\sigma, b_{1-\sigma}), \sigma) \equiv do\ \{\\
&\quad (\alpha, \tau) \leftarrow I;\\
&\quad x_\sigma \leftarrow S(\alpha);\\
&\quad y_{1-\sigma} \leftarrow S(\alpha);\\
&\quad let\ y_\sigma = F(\alpha, x_\sigma);\\
&\quad let\ x_\sigma = F^{-1}(\alpha, \tau, y_\sigma);\\
&\quad let\ x_{1-\sigma} = F^{-1}(\alpha, \tau, y_{1-\sigma});\\
&\quad let\ \beta_\sigma = B(\alpha, x_\sigma) \oplus b_\sigma;\\
&\quad let\ \beta_{1-\sigma} = B(\alpha, x_{1-\sigma}) \oplus b_{1-\sigma};\\
&\quad return(\_, if\ \sigma\ then\ B(\alpha, x_{1-\sigma}) \oplus \beta_{1-\sigma}\ else\ B(\alpha, x_\sigma) \oplus \beta_\sigma)\}
\end{aligned} \tag{7.18}$$

Using this definition and the functionality, for OT^1_2, given in Equation 7.11 we show correctness of Protocol 1.

Theorem 13. *(in etp-ot)*
shows $protocol_{OT^1_2,ETP}((b_0, b_1), \sigma) = funct_{OT^1_2}((b_0, b_1), \sigma)$

We next show the protocol is secure. We consider each party in turn and construct an appropriate simulator.

PARTY 1 For party 1 the real view is as follows.

$$\begin{aligned}
&R_{1,OT^1_2,ETP}\ ((b_0, b_1), \sigma) \equiv do\ \{\\
&\quad (\alpha, \tau) \leftarrow I;\\
&\quad x_\sigma \leftarrow S(\alpha);\\
&\quad y_{1-\sigma} \leftarrow S(\alpha);\\
&\quad let\ y_\sigma = F(\alpha, x_\sigma);\\
&\quad return((b_0, b_1), if\ \sigma\ then\ y_{1-\sigma}\ else\ y_\sigma, if\ \sigma\ then\ y_\sigma\ else\ y_{1-\sigma}\}
\end{aligned} \tag{7.19}$$

The only part of the real view that uses the party 2's input is in the return statement, where σ decides between y_σ and $y_{1-\sigma}$. The difference between the two (y_σ and $y_{1-\sigma}$) is that one is a sample direct from S and the other a sample from S that has been permuted under F. The following simulator suffices.

$$\begin{array}{l}
S_{1,OT^1_2,ETP}(\sigma, b_\sigma) \equiv do\ \{ \\
\quad (\alpha, \tau) \leftarrow I; \\
\quad y_0 \leftarrow S(\alpha); \\
\quad y_1 \leftarrow S(\alpha); \\
\quad return((b_0, b_1), y_0, y_1)\}
\end{array} \tag{7.20}$$

Theorem 14. *(in etp-ot)*
shows *perfect-sec-*$P_{1,OT^1_2,ETP}((b_0, b_1), \sigma)$

Proof. We only have to show that applying F has no effect on the output distribution. The real view applies F to obtain y_σ whereas the simulated view does not. This is the result we proved in Lemma 32. After applying this lemma the proof is completed by considering the cases on σ. □

PARTY 2 The proof of security for party 2 is considerably more involved. We follow the proof method and structure from [48, Section 4.3].

The real view is constructed as follows.

$$\begin{array}{l}
R_{2,OT^1_2,ETP}\ ((b_0, b_1), \sigma) \equiv do\ \{ \\
\quad (\alpha, \tau) \leftarrow I; \\
\quad x_\sigma \leftarrow S(\alpha); \\
\quad y_{1-\sigma} \leftarrow S(\alpha); \\
\quad let\ y_\sigma = F(\alpha, x_\sigma); \\
\quad let\ x_\sigma = F^{-1}(\alpha, \tau, y_\sigma); \\
\quad let\ x_{1-\sigma} = F^{-1}(\alpha, \tau, y_{1-\sigma}); \\
\quad let\ \beta_\sigma = B(\alpha, x_\sigma) \oplus (if\ \sigma\ then\ b_1\ else\ b_0); \\
\quad let\ \beta_{1-\sigma} = B(\alpha, x_{1-\sigma}) \oplus (if\ \sigma\ then\ b_0\ else\ b_1); \\
\quad return(\sigma, \alpha, (\beta_\sigma, \beta_{1-\sigma}))\}
\end{array} \tag{7.21}$$

We claim the following simulator suffices to prove security.

$$\begin{array}{l}
S_{2,OT^1_2,ETP}(\sigma, b_\sigma) \equiv do\ \{ \\
\quad (\alpha, \tau) \leftarrow I; \\
\quad x_\sigma \leftarrow S(\alpha); \\
\quad y_{1-\sigma} \leftarrow S(\alpha); \\
\quad let\ x_{1-\sigma} = F^{-1}(\alpha, \tau, y_{1-\sigma}); \\
\quad let\ \beta_\sigma = B(\alpha, x_\sigma) \oplus b_\sigma; \\
\quad let\ \beta_{1-\sigma} = B(\alpha, x_{1-\sigma}); \\
\quad return(\sigma, \alpha, (\beta_\sigma, \beta_{1-\sigma}))\}
\end{array} \tag{7.22}$$

The difference between the real view and the simulator is that the simulator cannot construct $\beta_{1-\sigma}$ correctly as it does not know $b_{1-\sigma}$. In the concrete setting we show the advantage ($\mathit{adv}\text{-}P_{2,OT^1_2,ETP}$) is less than or equal to $2 \cdot \mathit{adv}_{HCP}$. When we instantiate the security parameter we will show that $\mathit{adv}\text{-}P_{2,OT^1_2,ETP}$ is negligible (as the HCP advantage is assumed to be negligible), we discuss this in more detail in section 7.3.1.4.

As in [48] we split the proof into cases on b_σ. For the case where $b_{1-\sigma} = \mathit{False}$ we have perfect security as the simulator is equal to the real view.

Lemma 33. *(in etp-ot)*
assumes $b_{1-\sigma} = \mathit{False}$
shows $R_{2,OT^1_2,ETP}((b_0, b_1), \sigma) = S_{2,OT^1_2,ETP}((b_0, b_1), \sigma)$

Proof. The result comes trivially by using the assumption. □

The important Corollary to this Lemma is that the advantage for party 2 in this case is less than $2 \cdot \mathit{adv}_{HCP}$.

Corollary 1. *(in etp-ot)*
assumes $b_{1-\sigma} = \mathit{False}$
shows $\mathit{adv}\text{-}P_{2,OT^1_2,ETP}((b_0, b_1), \sigma) \leq 2 \cdot \mathit{adv}_{HCP}(\mathcal{A}_{HCP}, \sigma, b_\sigma, D)$

Proof. We know the HCP advantage must be greater than or equal to zero, moreover we have that $\mathit{adv}\text{-}P_{2,OT^1_2,ETP} = 0$ by Lemma 33 thus the result follows. □

It is left to consider the case where $b_{1-\sigma} = \mathit{True}$. To show security here we construct an adversary, $\mathcal{A}_{HCP}$ (given in Equation 7.23), that breaks the HCP assumption if D (which is an input to $\mathcal{A}_{HCP}$) can distinguish the real and simulated views — in other words we show a reduction to the HCP assumption.

$$\begin{aligned} &\mathcal{A}_{HCP}(\mathcal{A}, \sigma, b_\sigma, D) \equiv \mathit{do}\ \{ \\ &\quad \beta_{1-\sigma} \leftarrow \mathit{coin}; \\ &\quad x_\sigma \leftarrow S(\alpha); \\ &\quad \mathit{let}\ \beta_\sigma = B(\alpha, x_\sigma) \oplus b_\sigma; \\ &\quad d \leftarrow D(\sigma, \alpha, \beta_\sigma, \beta_{1-\sigma}); \\ &\quad \mathit{return}(\mathit{if}\ d\ \mathit{then}\ \beta_{1-\sigma}\ \mathit{else}\ \neg\beta_{1-\sigma})\} \end{aligned} \tag{7.23}$$

The advantage this adversary has against the HCP assumption is the same as the advantage a distinguisher has in distinguishing the real and simulated views.

Lemma 34. *(in etp-ot)*
assumes $b_{1-\sigma} = \mathit{True}$
and $\forall a.\ \mathit{lossless\text{-}spmf}(D(a))$
shows $\mathit{adv}\text{-}P_{2,OT^1_2,ETP}((b_0, b_1), \sigma, D) = 2 \cdot \mathit{adv}_{HCP}(\mathcal{A}_{HCP}, \sigma, b_\sigma, D)$

Proof. The proof is technical and involved. We formally define a number of intermediate probabilistic programs that bridge the gap between the two sides of the equality incrementally. Our formal proof follows the overall structure of Lindell's proof in [48], which consists of a chain of probabilities: starting with the probability the adversary ($\mathcal{A}_{HCP}$) has of guessing the HCP — in our language this is adv_{HCP} — and eventually showing this is equal to the probability of the distinguisher distinguishing the real and simulated views — in our language this is $adv\text{-}P_{2,OT^1_2,ETP}$. For each intermediate step Lindell takes, we define an intermediate probabilistic program.

We highlight one proof step that was formally more difficult to reason about than the others. This is the first step of the proof in [48, first equality of p14] where we are required to split the probability of $\mathcal{A}_{HCP}$ winning the HCP game into two cases, dependent on the coin flip $\mathcal{A}_{HCP}$ makes ($\beta_{1-\sigma}$, in Equation 7.23).

Informally we show that (the following is equivalent to what Lindell writes in [48]),

$$\begin{aligned} Pr[\mathcal{A}_{HCP}\ wins\ HCP\ game] = \frac{1}{2} \cdot Pr[\mathcal{A}_{HCP}\ wins\ HCP\ game \mid \beta_{1-\sigma} = HCP] \\ + \frac{1}{2} \cdot Pr[\mathcal{A}_{HCP}\ wins\ HCP\ game \mid \beta_{1-\sigma} \neq HCP]. \end{aligned}$$

On paper this is a step is a simple conditional probability argument. To prove this formally we define two intermediate probabilistic programs: $HCP\text{-}game_{true}$ and $HCP\text{-}game_{false}$.

```
$HCP\text{-}game_{true}(\sigma, b_\sigma) = do\ \{$
  $(\alpha, \tau) \leftarrow I;$
  $x_\sigma \leftarrow S(\alpha);$
  $x_{1-\sigma} \leftarrow S(\alpha);$
  $let\ \beta_\sigma = B(\alpha, x_\sigma) \oplus b_\sigma;$
  $\boxed{let\ \beta_{1-\sigma} = B(\alpha, (F^{-1}(\alpha, \tau, x_{1-\sigma}));}$
  $d \leftarrow D(\sigma, \alpha, \beta_\sigma, \beta_{1-\sigma});$
  $let\ b' = (if\ d\ then\ \beta_{1-\sigma}\ else\ \neg\beta_{1-\sigma});$
  $let\ b = B(\alpha, (F^{-1}(\alpha, \tau, x_{1-\sigma})));$
  $return(b = b')\}$
```

```
$HCP\text{-}game_{false}(\sigma, b_\sigma) = do\ \{$
  $(\alpha, \tau) \leftarrow I;$
  $x_\sigma \leftarrow S(\alpha);$
  $x_{1-\sigma} \leftarrow S(\alpha);$
  $let\ \beta_\sigma = B(\alpha, x_\sigma) \oplus b_\sigma;$
  $\boxed{let\ \beta_{1-\sigma} = \neg B(\alpha, (F^{-1}(\alpha, \tau, x_{1-\sigma})));}$
  $d \leftarrow D(\sigma, \alpha, \beta_\sigma, \beta_{1-\sigma});$
  $let\ b' = (if\ d\ then\ \beta_{1-\sigma}\ else\ \neg\beta_{1-\sigma});$
  $let\ b = B(\alpha, (F^{-1}(\alpha, \tau, x_{1-\sigma})));$
  $return(b = b')\}$
```

The difference in the two programs is subtle and highlighted in the above definitions.

The formal proof for this step is challenging as $\beta_{1-\sigma}$ is a bound variable inside the probabilistic program that defines $\mathcal{A}_{HCP}$. Accessing and dealing with this requires some underlying probability (in particular results on integration) theory formalised in Isabelle. More precisely, we are required to prove that extracting the sample from the probabilistic program is legitimate so the cases can be reasoned about. The formal statement of this step is as follows:

$$\begin{aligned} &\mathcal{P}[HCP_{game}(\mathcal{A}, \sigma, b_\sigma, D) = True] = \\ &\quad \frac{1}{2} \cdot \mathcal{P}[HCP\text{-}game_{true}(\sigma, b_\sigma) = True] + \frac{1}{2} \cdot \mathcal{P}[HCP\text{-}game_{false}(\sigma, b_\sigma) = True] \end{aligned} \tag{7.24}$$

The rest of the formal proof follows the chain of equalities given in [48] in a similar manner as described above. We require the losslessness assumption on D because if the distinguisher does not produce an output we cannot reason about any of the probabilities in the result. □

Using Corollary 1 and Lemma 34 we bound the advantage for party 2.

Theorem 15. *(in etp-ot)*
assumes $\forall a.\ \mathit{lossless\text{-}spmf}(D(a))$
and $\mathit{adv}_{HCP}(\mathcal{A}_{HCP}, m_2, b_\sigma, D) \leq \mathit{HCP}_{adv}$
shows $\mathit{adv\text{-}P}_{2,OT^1_2,ETP}((b_0, b_1), \sigma, D) \leq 2 \cdot \mathit{HCP}_{adv}$

Together Theorems 14 and 15 show Protocol 1 is secure in the concrete setting. For party 1 we showed perfect security and for party 2 we reduced security to the HCP assumption.

Next we show how we instantiate this general security result for the RSA collection of permutations, which form an ETP.

7.3.1.3 *Instantiating for RSA*

The result proven in section 7.3.1.2 is general, assuming only that an ETP exists. To instantiate it we only need to prove a collection of functions satisfies the assumptions of the locale *etp-base* (Equation 7.15) are satisfied.

It is known that the RSA collection of functions provides an ETP (see [37, Section 2.4.4.2] together with [38, Section C.1]). Here we formalise this RSA collection and instantiate it for Protocol 1.

To formalise the RSA collection we fix as a parameter a set of primes (*prime-set* :: *nat set*), in a locale, along with the assumed HCP, B, which we assume to exist. The set of primes is required to sample the prime factors of the RSA modulus.

$$\begin{aligned}
&\textbf{locale } \mathit{rsa\text{-}base} = \\
&\quad \textbf{fixes } \mathit{prime\text{-}set} :: \mathit{nat\ set} \\
&\qquad \textbf{and } B :: {}'\mathit{index} \Rightarrow \mathit{nat} \Rightarrow \mathit{bool} \\
&\quad \textbf{assumes } \mathit{prime\text{-}set} \subseteq \{x.\ \mathit{prime}(x) \wedge x > 2\} \\
&\qquad \textbf{and } \mathit{finite}(\mathit{prime\text{-}set}) \\
&\qquad \textbf{and } \mathit{card}(\mathit{prime\text{-}set}) > 2
\end{aligned} \tag{7.25}$$

The assumptions ensure the the set of primes we fix have the desired properties. The set *prime-set* can be chosen to not contain any *weak* primes for which there are known attacks on RSA.

To define the algorithms that make up the RSA ETP we define a set of coprimes (to the RSA modulus), we will sample from this set in the algorithm I_{RSA}.

Definition 24 (Coprime set)**.**

$$\mathit{coprime\text{-}set}(N) = \{x.\ \mathit{coprime}(x, N) \wedge x > 1 \wedge x < N\}$$

For ease we also define the uniform sampling from this set and the set of primes and from the set of primes modulo a prime.[6]

[6] When sampling the modulus for the RSA function we require two distinct primes.

Definition 25 (uniform samples for RSA ETP)**.**

$$\textit{sample-coprime} = \textit{uniform}(\textit{coprime-set}(N))$$

$$\textit{sample-prime} = \textit{uniform}(\textit{prime-set})$$

$$\textit{sample-prime-excl}(P) = \textit{uniform}(\textit{prime-set} - P)$$

The first component of the RSA ETP to define is I_{RSA}. The algorithm I_{RSA} uniformly selects two distinct primes, P and Q, and an integer e such that e is coprime to $(P-1)\cdot(Q-1)$ so e is uniformly sampled among all admissible options. The index of I_{RSA} is (N, e) where $N = P \cdot Q$ and the trapdoor $inv_{N'}(e)$ which is the inverse of e in the field of size $N' = (P-1)\cdot(Q-1)$.

$$\begin{aligned} & I_{RSA} \equiv do\ \{ \\ & \quad P \leftarrow \textit{sample-prime}; \\ & \quad Q \leftarrow \textit{sample-prime-excl}(P); \\ & \quad let\ N = P \cdot Q; \\ & \quad let\ N' = (P-1)\cdot(Q-1); \\ & \quad e \leftarrow \textit{sample-coprime}(N'); \\ & \quad let\ d = inv_{N'}(e); \\ & \quad return((N, e), d)\} \end{aligned} \tag{7.26}$$

The domain and range are equal,

$$domain_{RSA}(N, e) = \{.. < N\} \tag{7.27}$$

$$range_{RSA}(N, e) = \{.. < N\} \tag{7.28}$$

and the RSA function and its (trapdoor) inverse is defined as follows:

$$F_{RSA}((N, e), x) = x^e \ mod\ N. \tag{7.29}$$

$$F_{RSA}^{-1}((N, e), d, y) = y^d \ mod\ N \tag{7.30}$$

The definition is contextual on N, e and d being sampled by I_{RSA}.

Having defined the ETP for RSA we must show it is in fact an ETP with respect to our *etp* locale (Equation 7.15). Here we can use Isabelle's locale structure and show our RSA construction is a sublocale of *etp*. This means all results from the original locale are now valid and available in the new sublocale.

$$\textbf{sublocale}\ etp_{RSA} :\ \textit{etp-base}\ I_{RSA}\ domain_{RSA}\ range_{RSA}\ F_{RSA}\ F_{RSA}^{-1} \tag{7.31}$$

For the sublocale command to be valid we must prove our new construction meets the assumptions made in the *etp-base* locale: that it is we must prove the domain equals the range, the range is finite, the range is non-empty, that f_{RSA} is a bijection, applying

F_{RSA}^{-1} to F_{RSA} returns the original input and that I_{RSA} is lossless. The two non-trivial assumptions are to prove is that f_{RSA} is a bijection and that the inverse is correct. We first consider the proof of the bijection.

It is often the case when formalising paper proofs that detailed proofs of "obvious" are in fact difficult to reconstruct and it is hard to find proofs in the literature with sufficient detail to be useful in the formalisation.

Formally we must prove the following.

Lemma 35. *(in rsa-base)*
assumes $prime(P)$
and $prime(Q)$
and $P \neq Q$
and $coprime(e, (P-1) \cdot (Q-1))$
and $x, y < P \cdot Q$
and $x^e \ \ mod \ (P \cdot Q) = y^e \ \ mod \ (P \cdot Q)$
shows $x = y.$

Proof. From the assumption that $x^e \ mod(P \cdot Q) = y^e \ mod(P \cdot Q)$ we have that $x^e \ mod \ N = y^e \ mod \ N$. In turn this implies that N divides $x^e - y^e$ meaning $x^e \ mod \ s = y^e \ mod \ s$ where s is any prime factor of N — in particular we have $x^e \ mod \ P = y^e \ mod \ P$ and $x^e \ mod \ Q = y^e \ mod \ Q$.

From the assumption that e and $(P-1) \cdot (Q-1)$ are coprime we know there exists a d such that for some $k \geq 0$ we have $e \cdot d = 1 + k \cdot (P-1)$. This comes because from the assumption we know e and $P-1$ are coprime and thus the inverse of e exists modulo $P-1$. If we take $x^e \ mod \ P = y^e \ mod \ P$ to the power d we get

$$x^{1+k \cdot (P-1)} \ mod \ P = y^{1+k \cdot (P-1)} \ mod \ P.$$

Using Fermat's Little Theorem (FLT) we get $x \ mod \ P = y \ mod \ Q$. While reasoning about this on paper takes a little thinking, in Isabelle we can prove it quite easily by induction on k, however we comment that we needed to adapt the formalised result of FLT slightly from the version provided in the Isabelle library. In particular we needed the result where the exponent is p and not $p-1$, the adapted lemma was simple to prove.

From $x \ mod \ P = y \ mod \ Q$ we know that P divides $x - y$, and analogously we have that Q divides $x - y$. We know $P \neq Q$ and therefore we know $P \cdot Q$ divides $x - y$, that is $x \ mod \ (P \cdot Q) = y \ mod \ (P \cdot Q)$.

Both x and y are less than $P \cdot Q$ meaning we know $x = y$. □

Using this result we prove the f_{RSA} is a bijection.

Corollary 2. *(in rsa-base)*
assumes $(\alpha, \tau) \in set\text{-}spmf(I)$
shows $bij\text{-}betw(f_{RSA}(\alpha), domain(\alpha), range(\alpha))$

We next prove the correctness of the inverse function by first considering the follwoing lemma.

Lemma 36. *(in rsa-base)*
assumes $prime(P)$
and $prime(Q)$
and $P \neq Q$
and $coprime(e, (P-1) \cdot (Q-1))$
and $e > 1$
and $d > 1$
and $x^e \ \ mod \ (P \cdot Q) = y^e \ \ mod \ (P \cdot Q)$
defines $d = inv_{(P-1)\cdot(Q-1)}(e)$
shows $((x^e \ mod \ (P \cdot Q))^d) \ mod \ (P \cdot Q) = x \ mod \ (P \cdot Q).$

Proof. We treat the case where $x = 0 \vee x = 1$ first as this causes difficulties later otherwise. For the remaining cases we show that $[x^{e \cdot d} - x = 0] \ mod \ P$ and $[x^{e \cdot d} - x = 0] \ mod \ Q$ which means that P and Q both divide $(x^e)^d - x$ meaning that $P \cdot Q$ divides $(x^e)^d - x$ (as P and Q are both primes). Our result comes easily from this. □

As an immediate Corollary to Lemma 36 we prove the assumption from the *etp-base* locale. In the statement below we have unfolded the definitions for clarity.

Corollary 3. *(in rsa-base)*
assumes $((N, e), d) \in \textit{set-spmf}(I)$
shows $((x^e \ mod \ (P \cdot Q))^d) \ mod \ (P \cdot Q) = x \ mod \ (P \cdot Q).$

Proof. The assumptions in Lemma 36 are all properties of $((N, e), d) \in \textit{set-spmf}(I)$, meaning they are all met by the assumption here. □

Using these results we can prove the sublocale given in Equation 7.31 is valid, that is the assumptions are met by the parameters defined for the RSA instance.

The security and correctness proofs now come for free (one line of proof each; unfolding definitions). The statements can be seen below.

Theorem 16. *(in rsa-base)*
shows $protocol_{OT^1_2, RSA}((b_0, b_1), \sigma) = funct_{OT^1_2}((b_0, b_1), \sigma)$

Theorem 17. *(in rsa-base)*
shows $\textit{perfect-sec-P}_{1, OT^1_2, RSA}((b_0, b_1), \sigma)$

Theorem 18. *(in rsa-base)*
assumes $\forall a. \ \textit{lossless-spmf}(D(a))$
and $adv_{HCP}(\mathcal{A}_{HCP}, m_2, b_\sigma, D) \leq HCP_{adv}$
shows $adv\_P_{2, OT^1_2, RSA}(((b_0, b_1), \sigma), D) \leq 2 \cdot HCP_{adv}$

Finally, this has shown that, assuming an HCP exists for RSA, we can securely compute OT^1_2 in the semi-honest model using the ETP obtained from the RSA function.

This proof highlights the strengths of Isabelle's module system. Initially we completed the proof for the RSA instantiation in full from scratch — replicating the proof from section 7.3.1.2. Subsequent leveraging of the module system, that we have detailed here, allowed us to halve the proof effort (in lines of proof).

Anyone wishing to prove further instantiations only needs to define the ETP and prove that the assumptions given in the *etp-base* locale are valid.

7.3.1.4 *The RSA instantiation in the asymptotic setting*

In this section we prove security in the asymptotic setting for the RSA instantiation.

Reasoning over the security parameter in the asymptotic setting allows a closer equivalence to the pen and paper security properties. One area where this is realised is in the ability to more accurately define hardness assumptions. For example in Theorem 17 we could only bound the advantage of party 2 by the HCP advantage. While this implies security we would like to explicitly make the assumption, as a function of an increasing security parameter, that the HCP advantage is negligible, and thus so is the advantage for party 2.

In the case of the RSA collection of permutations the security parameter will determine the set of primes that is sampled from in I_{RSA} — intuitively the larger the security parameter the larger the primes that will be chosen. Consequently we introduce the security parameter as an input to the set of primes we have fixed.

$$\begin{aligned} &\textbf{locale } \textit{etp-rsa-asym} = \\ &\quad \textbf{fixes } \textit{prime-set} :: \textit{nat} \Rightarrow \textit{nat set} \\ &\qquad \textbf{and } B :: \textit{'index} \Rightarrow \textit{nat} \Rightarrow \textit{bool} \\ &\quad \textbf{assumes } \textit{rsa-base}(\textit{prime-set}(n)) \end{aligned} \tag{7.32}$$

To realise this we import the concrete setting parametrically for all n in the *etp-rsa-asym* locale. Now all algorithms depend explicitly on the security parameter. Moreover, due to Isabelle's module structure we are able to use results proven in the concrete setting in our newly constructed asymptotic setting.

As in the previous section, the concrete setting can only be used once it has been proven that the import is valid (using the sublocale command). Once this is done we prove the security results in the asymptotic setting. First we show correctness is still valid and then that security holds.

Theorem 19. *(in etp-rsa-asym)*
shows $\textit{protocol}_{OT^1_2,RSA}(n,(b_0,b_1),\sigma) = \textit{funct}_{OT^1_2}((b_0,b_1),\sigma)$

The security parameter only appears as inputs to functions where it is used. Equation 7.11 shows that the security parameter is never required to define $\textit{funct}_{OT^1_2}$, and thus n does not appear as an input. Security is shown by the following theorems.

Theorem 20. *(in etp-rsa-asym)*
shows $\textit{perfect-sec-}P_{1,OT^1_2,RSA}(n,(b_0,b_1),\sigma)$

Theorem 21. *(in etp-rsa-asym)*
assumes $\textit{negligible}(\lambda\ n.\ \textit{adv}_{HCP}(n,\mathcal{A}_{HCP},b_\sigma,D))$
shows $\textit{negligible}(\lambda\ n.\ \textit{adv-}P_{2,OT^1_2,RSA}(n,(b_0,b_1),\sigma,D)$

In particular Theorem 21 shows the advantage for party 2 is negligible.

Our case study of the RSA ETP shows how our general proof can be instantiated without having to consider any secuirty properties of the underlying protocol. Any other ETP could be instantiated in the same way.

$$\begin{array}{ccc}
P_1 \text{ (Sender)} & & P_2 \text{ (Receiver)} \\
(m_0, m_1) \in G^2 & & \sigma \in \{0,1\} \\
 & & a,b \xleftarrow{\$} \mathbb{Z}_{|G|} \\
 & & c_\sigma = a.b,\ c_{1-\sigma} \xleftarrow{\$} \mathbb{Z}_{|G|} \\
 & & x \leftarrow g^a,\ y \leftarrow g^b \\
 & & z_0 \leftarrow g^{c_0},\ z_1 \leftarrow g^{c_1} \\
r_0, s_0 \xleftarrow{\$} \mathbb{Z}^*_{|G|} & \xleftarrow{(x,y,z_0,z_1)} & \\
r_1, s_1 \xleftarrow{\$} \mathbb{Z}^*_{|G|} & & \\
w_0 \leftarrow x^{s_0} \cdot g^{r_0} & & \\
z'_0 \leftarrow z_0^{s_0} \cdot y^{r_0} & & \\
w_1 \leftarrow x^{s_1} \cdot g^{r_1} & & \\
z'_1 \leftarrow z_1^{s_1} \cdot y^{r_1} & & \\
enc_{m_0} \leftarrow z'_0 \cdot m_0 & & \\
enc_{m_1} \leftarrow z'_1 \cdot m_1 & & \\
CT_0 = (w_0, enc_{m_0}) & & \\
CT_1 = (w_1, enc_{m_1}) & \xrightarrow{(CT_0,CT_1)} & CT_0, CT_1 \\
 & & \text{outputs } \frac{enc_{m_\sigma}}{w_\sigma^b}
\end{array}$$

Figure 7.2: The Naor-Pinkas OT protocol.

7.3.2 *Naor-Pinkas OT^1_2*

In this section we formally prove the Naor-Pinkas OT^1_2 secure.

The Naor-Pinkas OT protocol [57] uses a cyclic group G such that $|G|$ is a prime and generator g. The message space is taken as the group G meaning the party 1's messages are elements of G — $(m_0, m_1) \in G^2$. It is assumed that the DDH assumption holds for G. The Decisional Diffie Hellman (DDH) assumption [33] is a computational hardness assumption on cyclic groups. Informally, the assumption states that given g^a and g^b, where a and b are uniform samples from $\mathbb{Z}_{|G|}$, the group element $g^{a \cdot b}$ looks like a random element from G. A triple of the form $(g^a, g^b, g^{a \cdot b})$ is called a DDH triple.

The protocol is given in Figure 7.2. In the protocol the Sender holds the messages $(m_0, m_1) \in G^2$ and the Receiver holds its choice bit $\sigma \in \{0,1\}$. Here G is a cyclic group for which the Diffie Hellman assumption holds. The Receiver uniformly samples a, b and $c_{1-\sigma}$ from $\mathbb{Z}_q$ and computes $c_\sigma = a \cdot b$. They then compute $x = g^a, y = g^b, z_0 = g^{c_0}$ and $z_1 = g^{c_1}$. The Sender then computes encryptions for both m_0 and m_1 and sends them to the Receiver. As is common with OT^1_2 only one of these encryptions is valid and can be decrypted by the Receiver. In this case one is valid because $c_\sigma = a \cdot b$ and thus a DDH triple is formed with x and y.

Correctness of the protocol can be seen by considering the cases on σ.

For the Sender we have perfect security, intuitively the Receiver is only able to decrypt m_σ as the corresponding ciphertext is a valid ElGamal ciphertext whereas the other ciphertext is random, meaning nothing can be learnt of $m_{1-\sigma}$. Security for the Receiver is proven with a reduction to the DDH assumption. In particular the Sender can only learn σ if they can break the DDH assumption.

7.3.2.1 *The Formal Proof*

DDH ASSUMPTION The DDH assumption was formalised by Lochbihler et al. in [50, 53]. Here we document their formalisation for completeness. The DDH assumption is a computational hardness assumption concerning cyclic groups. It is used in the security of the ElGamal encryption [36], but is also used in the Cramer-Shoup cryptosystem [30] as well as many others. In our work the security of the Receiver (Party 2) is reduced to the DDH assumption in the Naor Pinkas OT_2^1 protocol (Section 7.3.2).

Informally, the assumption states that given g^a and g^b, where a and b are uniform samples from $\mathbb{Z}_{|G|}$, the group element $g^{a \cdot b}$ looks like a random element from G. A triple of the form $(g^a, g^b, g^{a \cdot b})$ is called a DDH triple, and a tuple of the form (g^a, g^b, g^c) is called a non-tuple. It is formalised by defining two games. The first provides the adversary with a DDH tuple as follows,

$$\begin{aligned} & \mathit{ddh}\text{-}0(\mathcal{A}) = \mathit{do}\ \{ \\ & \quad x \leftarrow \mathit{uniform}(|G|); \\ & \quad y \leftarrow \mathit{uniform}(|G|); \\ & \quad \mathcal{A}(g^x, g^y, g^{x \cdot y})\} \end{aligned} \tag{7.33}$$

and a second game where the adversary is given a non tuple as follows,

$$\begin{aligned} & \mathit{ddh}\text{-}1(\mathcal{A}) = \mathit{do}\ \{ \\ & \quad x \leftarrow \mathit{uniform}(|G|); \\ & \quad y \leftarrow \mathit{uniform}(|G|); \\ & \quad z \leftarrow \mathit{uniform}(|G|); \\ & \quad \mathcal{A}(g^x, g^y, g^z)\} \end{aligned} \tag{7.34}$$

The advantage an adversary has of beating the DDH assumption is defined as the probability it has of telling the games apart.

Definition 26 (DDH advantage)**.**

$$\mathit{ddh}\text{-}\mathit{adv}(\mathcal{A}) = |\mathcal{P}[\mathit{ddh}\text{-}0(\mathcal{A}) = \mathit{True}] - \mathcal{P}[\mathit{ddh}\text{-}1(\mathcal{A}) = \mathit{True}]|$$

PROVING SECURITY The protocol is deterministic, so we use the simpler definitions of security given in section 7.2.1. First we consider correctness. As usual, we define the protocol in a probabilistic program and compare it to the functionality. To define the protocol we define what it means to sample from $\mathbb{Z}^*_{|G|}$, the field of units, we make the following simple definition:

Definition 27.

$$\mathit{samp}\text{-}\mathit{uniform}\text{-}\mathit{units}(q) = \mathit{uniform}(\{\cdots < q\} - \{0\})$$

The definition is contextual on q being a prime.

We construct a base locale to work in. Here we fix the cyclic group.

$$\begin{aligned}
&\textbf{locale } \textit{np-base} = \\
&\quad \textbf{fixes } G :: \textit{'grp cyclic-group} \\
&\quad \textbf{assumes } |G| > 0 \\
&\qquad \textbf{and } \textit{prime}(|G|)
\end{aligned} \tag{7.35}$$

As we are working with a cyclic group structure we also work in a locale where we inherit the cyclic group properties as follows.

$$\textbf{locale } \textit{np} = \textit{np-base} + \textit{cyclic-group}(G) \tag{7.36}$$

We define the protocol below and then prove correctness.

$$\begin{aligned}
&\textit{protocol}_{NP}((m_0, m_1), \sigma) = \textit{do } \{ \\
&\quad a \leftarrow \textit{samp-uniform}(|G|); \\
&\quad b \leftarrow \textit{samp-uniform}(|G|); \\
&\quad \textit{let } c_\sigma = (a \cdot b); \\
&\quad c_\sigma \leftarrow \textit{samp-uniform}(|G|); \\
&\quad r_0 \leftarrow \textit{samp-uniform-units}(|G|); \\
&\quad s_0 \leftarrow \textit{samp-uniform-units}(|G|); \\
&\quad \textit{let } w_0 = (g^a)^{s_0} \otimes g^{r_0}; \\
&\quad \textit{let } z_0' = (g^{\textit{if } \sigma \textit{ then } c_{1-\sigma} \textit{ else } c_\sigma})^{s_0} \otimes (g^b)^{r_0}; \\
&\quad r_1 \leftarrow \textit{samp-uniform-units}(|G|); \\
&\quad s_1 \leftarrow \textit{samp-uniform-units}(|G|); \\
&\quad \textit{let } w_1 = (g^a)^{s_1} \otimes g^{r_1}; \\
&\quad \textit{let } z_1' = (g^{\textit{if } \sigma \textit{ then } c_\sigma \textit{ else } c_{1-\sigma}})^{s_1} \otimes (g^b)^{r_1}; \\
&\quad \textit{let } enc_{m_0} = z_0' \otimes m_0; \\
&\quad \textit{let } enc_{m_1} = z_1' \otimes m_1; \\
&\quad \textit{let } out_2 = (\textit{if } \sigma \textit{ then } enc_{m_1} \otimes inv(w_1^b) \textit{ else } enc_{m_0} \otimes inv(w_0^b)); \\
&\quad \textit{return}((), out_2)\}
\end{aligned} \tag{7.37}$$

Theorem 22. *(in np)*

assumes $m_0 \in \textit{carrier}(G)$

and $m_1 \in \textit{carrier}(G)$

shows $\textit{protocol}_{NP}((m_0, m_1), \sigma) = \textit{funct}_{OT^1_2}((m_0, m_1), \sigma)$

Proof. To complete this proof Isabelle needs to be told how to rewrite identities in the group. We proved the following before proving correctness, assuming $m \in \textit{carrier}(G)$:

$$g^{a \cdot b} \otimes (g^b)^{r_1} \otimes m \otimes inv((g^a)^{s_1} \otimes g^{r_1})^b = m.$$

Once Isabelle rewrites this equality the proof follows easily. Writing the intermediary steps in just the right form help the automatic procedures find proofs more quickly. □

Next we prove security for the Naor-Pinkas OT^1_2, considering each party in turn.

PARTY 1 The real view for party 1 is defined as follows.

$$\begin{aligned}
&R_1((m_0, m_1), \sigma) = do\ \{\\
&\quad a \leftarrow \textit{samp-uniform}(|G|);\\
&\quad b \leftarrow \textit{samp-uniform}(|G|);\\
&\quad let\ c_\sigma = a \cdot b;\\
&\quad c_{1-\sigma} \leftarrow \textit{samp-uniform}(|G|);\\
&\quad return((m_0, m_1), g^a, g^b, if\ \sigma\ then\ g^{c_{1-\sigma}}\ else\ g^{c_\sigma},\ if\ \sigma\ then\ g^{c_\sigma}\ else\ g^{c_{1-\sigma}})\}
\end{aligned} \tag{7.38}$$

The intuition behind constructing the simulator is to notice that the output (last four elements) of the real view is a DDH tuple, with one extra random element ($g^{c_{1-\sigma}}$). We know that the DDH relation is a hard problem, thus we can show a reduction to it. In particular the simulator can output the final two elements of the output of party 1 in either order, we must choose one.

$$\begin{aligned}
&S_1((m_0, m_1), out_1) = do\ \{\\
&\quad a \leftarrow \textit{samp-uniform}(|G|);\\
&\quad b \leftarrow \textit{samp-uniform}(|G|);\\
&\quad c \leftarrow \textit{samp-uniform}(|G|);\\
&\quad return((m_0, m_1), g^a, g^b, g^c, g^{a \cdot b})\}
\end{aligned} \tag{7.39}$$

To prove security we show a reduction to the DDH assumption.

Theorem 23. *(in np)*
shows $\textit{adv-}P_{1,NP}((m_0, m_1), \sigma, D) \leq$
$\textit{ddh-adv}(\mathcal{A}_{int_1}, D, (m_0, m_1)) + \textit{ddh-adv}(\mathcal{A}_{int_2}, D, (m_0, m_1))$

where $\mathcal{A}_{int_1}$ and $\mathcal{A}_{int_2}$ are the intermediate adversaries we use against the DDH assumption.

Proof. For the case $\sigma = True$ the views are equal, we show this implies the advantage is zero and thus the inequality is satisfied. The case where $\sigma = False$ is more complex. The real and simulated views differ in the last two elements of the output. The simulator outputs $(g^c, g^{a \cdot b})$ and the real view outputs $(g^{a \cdot b}, g^c)$. We show the two outputs are indistinguishable by two reduction steps to the DDH assumption. We first show $((m_0, m_1), g^a, g^b, g^c, g^{a \cdot b})$ (the simulated view) is indistinguishable from $((m_0, m_1), g^a, g^b, g^c, g^d)$ (where all the exponents are uniformly sampled) by showing a reduction to the DDH assumption and then that $((m_0, m_1), g^a, g^b, g^c, g^d)$ and $((m_0, m_1), g^a, g^b, g^{a \cdot b}, g^d)$ are indistinguishable by another reduction to the DDH assumption. $\mathcal{A}_{int_1}$ and $\mathcal{A}_{int_2}$ are the adversaries we use to show the reductions. They are constructed as follows.

$$\begin{aligned}
&\mathcal{A}_{int_1}(\mathcal{A}, (m_0, m_1), a, b, c) = do\ \{\\
&\quad c' \leftarrow \textit{samp-uniform}(|G|);\\
&\quad \mathcal{A}((m_0, m_1), a, b, c, g^{c'})\}
\end{aligned}
\qquad
\begin{aligned}
&\mathcal{A}_{int_2}(\mathcal{A}, (m_0, m_1), a, b, c) = do\ \{\\
&\quad c' \leftarrow \textit{samp-uniform}(|G|);\\
&\quad \mathcal{A}((m_0, m_1), a, b, g^{c'}, c)\}
\end{aligned}$$

Together the adversaries interact with the DDH assumption to produce the required bound. □

PARTY 2 The real view for party 2 is given below.

$$
\begin{aligned}
&R_2((m_0, m_1), \sigma) = \mathit{do}\ \{\\
&\quad a \leftarrow \mathit{samp\text{-}uniform}(|G|);\\
&\quad b \leftarrow \mathit{samp\text{-}uniform}(|G|);\\
&\quad \mathit{let}\ c_\sigma = (a \cdot b);\\
&\quad c_{1-\sigma} \leftarrow \mathit{samp\text{-}uniform}(|G|);\\
&\quad r_0 \leftarrow \mathit{samp\text{-}uniform\text{-}units}(|G|);\\
&\quad s_0 \leftarrow \mathit{samp\text{-}uniform\text{-}units}(|G|);\\
&\quad \mathit{let}\ w_0 = (g^a)^{s_0} \otimes g^{r_0};\\
&\quad \mathit{let}\ z = (g^{c_{1-\sigma}})^{s_0} \otimes (g^b)^{r_0};\\
&\quad r_1 \leftarrow \mathit{samp\text{-}uniform\text{-}units}(|G|);\\
&\quad s_1 \leftarrow \mathit{samp\text{-}uniform\text{-}units}(|G|);\\
&\quad \mathit{let}\ w_1 = (g^a)^{s_1} \otimes g^{r_1};\\
&\quad \mathit{let}\ z' = (g^{c_\sigma})^{s_1} \otimes (g^b)^{r_1};\\
&\quad \mathit{let}\ enc_{m_\sigma} = z \otimes (\mathit{if}\ \sigma\ \mathit{then}\ m_0\ \mathit{else}\ m_1);\\
&\quad \mathit{let}\ enc_{m_{1-\sigma}} = z' \otimes (\mathit{if}\ \sigma\ \mathit{then}\ m_1\ \mathit{else}\ m_0);\\
&\quad \mathit{return}(\sigma, g^a, g^b, g^{c_{1-\sigma}}, w_0, enc_{m_\sigma}, w_1, enc_{m_{1-\sigma}})\}
\end{aligned}
\tag{7.40}
$$

The simulator for party 2 cannot correctly form enc_{m_σ} as it does not know $m_{1-\sigma}$ as this is the message not outputted to party 2 in the protocol. The simulator however can be constructed without this and be shown to equal the real view. The idea behind the construction of the simulator is that it can construct $enc_{m_\sigma} = g^s$ where s is a unique uniform sample.

$$\begin{aligned}
&S_2(\sigma, m) = do\ \{\\
&\quad a \leftarrow samp\text{-}uniform(|G|);\\
&\quad b \leftarrow samp\text{-}uniform(|G|);\\
&\quad let\ c_\sigma = (a \cdot b);\\
&\quad r_0 \leftarrow samp\text{-}uniform\text{-}units(|G|);\\
&\quad s_0 \leftarrow samp\text{-}uniform\text{-}units(|G|);\\
&\quad let\ w_0 = (g^a)^{s_0} \otimes g^{r_0};\\
&\quad r_1 \leftarrow samp\text{-}uniform\text{-}units(|G|);\\
&\quad s_1 \leftarrow samp\text{-}uniform\text{-}units(|G|);\\
&\quad let\ w_1 = (g^a)^{s_1} \otimes g^{r_1};\\
&\quad let\ z' = (g^{c_\sigma})^{s_1} \otimes (g^b)^{r_1};\\
&\quad s \leftarrow samp\text{-}uniform(|G|);\\
&\quad let\ enc_{m_\sigma} = g^s;\\
&\quad let\ enc_{m_{1-\sigma}} = z' \otimes m;\\
&\quad return(\sigma, g^a, g^b, g^{c_{1-\sigma}}, w_0, enc_{m_\sigma}, w_1, enc_{m_{1-\sigma}})\}
\end{aligned} \tag{7.41}$$

We show perfect security in the following theorem.

Theorem 24. *(in np)*
assumes $m_0 \in carrier(G)$
and $m_1 \in carrier(G)$
shows $perfect\text{-}sec\text{-}P_{2,NP}((m_0, m_1), \sigma)$

Proof. The difference between the simulator and the real view is that the simulator cannot form enc_{m_σ} in the correct way. We claim however that the distributions produced by enc_{m_σ} in the simulator and the real view are the same. In the real view $enc_{m_\sigma} = z \otimes (if\ \sigma\ then\ m_0\ else\ m_1)$ where $z = (g^{c_{1-\sigma}})^{s_0} \otimes (g^b)^{r_0}$. If we show z is a uniform sample from the group, that is independent of any of the other variables outputted, then we can apply the classic one time pad result for a cyclic group (proven in [50, 51]). Namely: *if* $c \in carrier(G)$ then we have

$$map(\lambda x.\ g^x \otimes c, samp\text{-}uniform(|G|)) = map(\lambda x.\ g^x, samp\text{-}uniform(|G|)) \tag{7.42}$$

We rewrite z as $z = g^s$ where $s = (b \cdot r_0 + c_{1-\sigma} \cdot s_0)\ mod\ (|G|)$. Thus our problem is reduced to showing s is a uniform sample. To show this we use the randomness from $c_{1-\sigma}$ and apply the one time pad lemma: Assume $coprime(x, |G|)$, then we have

$$map(\lambda b.\ y + x \cdot b)\ mod\ (|G|), samp\text{-}uniform(|G|)) = samp\text{-}uniform(|G|) \tag{7.43}$$

We know that the assumption holds as $c_{1-\sigma}$ is sampled from the non zero elements up to $|G|$ and we know $|G|$ is a prime. We must use the randomness from $c_{1-\sigma}$ and not any of the other variables as it is the only one not used again in the output of the real view. Having shown s is a uniform sample we apply Equation 7.42 to complete the proof, specifically to turn $enc_{m_\sigma} = z \otimes (if\ \sigma\ then\ m_0\ else\ m_1)$ into $enc_{m_\sigma} = z$. □

Together Theorems 24 and 23 show security for the Naor Pinkas OT^1_2 protocol.

DISCUSSION ABOUT MODELLING OT^1_2 We highlight one design choice of our modeling of OT^1_2 we feel is important to be transparent about. Both the OT^1_2 protocols we consider here (ETP OT^1_2 and the Naor-Pinkas OT^1_2) follow, what could be considered, the standard form for OT^1_2 protocols: the Receiver creates some randomness and sends it to the Sender to encrypt both messages. The randomness is constructed in such a way that the encryption of one of the Sender's messages can be decrypted (the message chosen by the Receiver) and the other cannot. Moreover the randomness sent by the Receiver will not allow the Sender to learn anything about the choice bit of the Receiver.

The randomness created by the Receiver for each message to be encrypted by the Sender must be different, otherwise they could decrypt both messages. In the ETP OT^1_2 this manifests itself in the requirement that y_σ and $y_{1-\sigma}$ must not be equal and in the NP OT^1_2 z_0 cannot equal z_1. We however choose not to capture this in our definitions of the protocols or views. We justify this by considering the ETP OT^1_2 protocol as an example: the statement $y_\sigma \neq y_{1-\sigma}$ holds with overwhelming probability. Specifically y_σ and $y_{1-\sigma}$ are uniformly distributed, so the property fails to hold with probability $\frac{1}{|X|}$ where X is the sampling space. In this case $X = range(\alpha)$ and so is exponentially large (otherwise the ETP is not a trapdoor permutation) thus we argue we are justified in not capturing the property in our proofs. Note the analogous argument holds for the Naor-Pinkas OT^1_2, here the sample space is $\mathbb{Z}_{|G|}$.

7.4 GMW

In this section we show how we prove security for the gates of the two party GMW protocol [38, 39]. To realise the GMW protocol we require OT^1_4 and secret sharing. Here we first show how we formalise a protocol that realises OT^1_4 (Section 7.4.1) and then how we formalise secret sharing schemes, and in particular the secret sharing scheme required for the GMW protocol (Section 7.4.3). Finally in Section 7.4.2 we show how we prove the two party GMW protocol secure.

To formalise the two party GMW protocol we consider the building blocks in a modular way. We assume results on OT^1_2 when proving results on the OT^1_4 protocol and then assume the results we prove regarding the OT^1_4 protocol when considering the GMW protocol. The assumptions are all local to the locale (module) in which we work for each protocol. The advantage of this is theories are more standalone, they make assumptions on the required primitive (OT^1_2 or OT^1_4) and then prove the desired results (regarding OT^1_4 or GMW).

7.4.1 *A protocol that realises OT^1_4*

We take the protocol that realises OT^1_k from [38, Section 7.3.3, p640] and adapt it for the case of OT^1_4. The functionality for OT^1_4 is defined as,

$$funct_{OT^1_4}((b_{0,0}, b_{0,1}, b_{1,0}, b_{1,1}), (c_0, c_1)) = return((), b_{c_0,c_1}). \tag{7.44}$$

The Receiver (R) chooses one of four messages held by the Sender (S).

Protocol 2. *S has inputs $(b_{0,0}, b_{0,1}, b_{1,0}, b_{1,1}) \in \{0,1\}^4$ and R has input $(c_0, c_1) \in \{0,1\}^2$.*

1. *S uniformly randomly samples* $S_a \xleftarrow{\$} \{0,1\}$ *for* $a \in \{0,...,5\}$.
2. *S calculates the following:*

$$\alpha_0 = S_0 \oplus S_2 \oplus b_{0,0}, \ \alpha_1 = S_0 \oplus S_3 \oplus b_{0,1}$$
$$\alpha_2 = S_1 \oplus S_4 \oplus b_{1,0}, \ \alpha_3 = S_1 \oplus S_5 \oplus b_{1,1}.$$

3. *S and R then run three* OT^1_2 *protocols together. That is they run,*

$$(\_, S_i) \leftarrow OT^1_2((S_0, S_1), c_0)$$
$$(\_, S_j) \leftarrow OT^1_2((S_2, S_3), c_1)$$
$$(\_, S_k) \leftarrow OT^1_2((S_4, S_5), c_1)$$

4. *R calculates*

$$b_{c_0,c_1} = S_i \oplus (\mathit{if}\ c_0\ \mathit{then}\ S_k\ \mathit{else}\ S_j)\ \oplus (\mathit{if}\ c_0\ \mathit{then}\ (\mathit{if}\ c_1\ \mathit{then}\ \alpha_3\ \mathit{else}\ \alpha_2)$$
$$\mathit{else}\ (\mathit{if}\ c_1\ \mathit{then}\ \alpha_1\ \mathit{else}\ \alpha_0)).$$

Correctness of the protocol comes from the assumption of correctness of the OT^1_2. Security comes from the masking of the messages sent from S to R in Step 2 and the security of the OT^1_2.

7.4.1.1 *Formalising the protocol and its security*

To prove security of Protocol 2 we first make assumptions on the underlying OT^1_2, we assume: correctness, perfect security for the Receiver and bound the advantage of party 1 by a real parameter. These are the security results of the Naor Pinkas OT^1_2 which is used in practical implementations of GMW. We fix these in the locale OT^1_4*-base* below.

$$
\begin{aligned}
&\textbf{locale}\ OT^1_4\textit{-base} = \\
&\quad \textbf{fixes}\ \mathit{protocol}_{OT^1_2}\ ::\ (\mathit{bool} \times \mathit{bool}) \Rightarrow \mathit{bool} \Rightarrow (\mathit{unit} \times \mathit{bool})\ \mathit{spmf} \\
&\qquad \textbf{and}\ R_{1,OT^1_2}\ ::\ (\mathit{bool} \times \mathit{bool}) \Rightarrow \mathit{bool} \Rightarrow \text{'}\mathit{view}_{1,OT^1_2}\ \mathit{spmf} \\
&\qquad \textbf{and}\ S_{1,OT^1_2}\ ::\ (\mathit{bool} \times \mathit{bool}) \Rightarrow \mathit{unit} \Rightarrow \text{'}\mathit{view}_{1,OT^1_2}\ \mathit{spmf} \\
&\qquad \textbf{and}\ R_{2,OT^1_2}\ ::\ (\mathit{bool} \times \mathit{bool}) \Rightarrow \mathit{bool} \Rightarrow \text{'}\mathit{view}_{2,OT^1_2}\ \mathit{spmf} \\
&\qquad \textbf{and}\ S_{2,OT^1_2}\ ::\ \mathit{bool} \Rightarrow \mathit{bool} \Rightarrow \text{'}\mathit{view}_{2,OT^1_2}\ \mathit{spmf} \\
&\qquad \textbf{and}\ P_1\mathit{adv}_{OT^1_2}\ ::\ \mathit{real} \\
&\quad \textbf{assumes}\ \mathit{protocol}_{OT^1_2}((m_0, m_1), \sigma) = \mathit{funct}_{OT^1_2}((m_0, m_1), \sigma) \\
&\qquad \textbf{and}\ \mathit{adv}\text{-}P_{1,OT^1_2}((m_0, m_1), \sigma, D) \leq P_1\mathit{adv}_{OT^1_2} \\
&\qquad \textbf{and}\ P_1\mathit{adv}_{OT^1_2} > 0 \\
&\qquad \textbf{and}\ \mathit{perfect}\text{-}\mathit{sec}\text{-}P_{2,OT^1_2}((m_0, m_1), \sigma)
\end{aligned}
\tag{7.45}
$$

To show correctness we define the probabilistic program $protocol_{OT^1_4}$ that provides the output distribution of Protocol 2 as $protocol_{OT^1_4}$.

$$\begin{aligned}
&protocol_{OT^1_4}((b_{0,0}, b_{0,1}, b_{1,0}, b_{1,1}), (c_0, c_1)) = do \ \{\\
&\quad S_0, S_1, S_2, S_3, S_4, S_5 \leftarrow coin;\\
&\quad S_1 \leftarrow coin;\\
&\quad S_2 \leftarrow coin;\\
&\quad S_3 \leftarrow coin;\\
&\quad S_4 \leftarrow coin;\\
&\quad S_5 \leftarrow coin;\\
&\quad let\ a_0 = S_0 \oplus S_2 \oplus m_{0,0};\\
&\quad let\ a_1 = S_0 \oplus S_3 \oplus m_{0,1};\\
&\quad let\ a_2 = S_1 \oplus S_4 \oplus m_{1,0};\\
&\quad let\ a_3 = S_1 \oplus S_5 \oplus m_{1,1};\\
&\quad (\_, S_i) \leftarrow protocol_{OT^1_2}((S_0, S_1), c_0);\\
&\quad (\_, S_j) \leftarrow protocol_{OT^1_2}((S_2, S_3), c_1);\\
&\quad (\_, S_k) \leftarrow protocol_{OT^1_2}((S_4, S_5), c_1);\\
&\quad return((), S_i \oplus (if\ c_0\ then\ S_k\ else\ S_j) \oplus\\
&\qquad\qquad (if\ c_0\ then\ ((if\ c_1\ then\ a_3\ else\ a_2))\ else\ (if\ c_1\ then\ a_1\ else\ a_0)))\}
\end{aligned} \tag{7.46}$$

Theorem 25 shows the correctness of Protocol 2. Here B and C represent the inputs for the Sender $(b_{0,0}, b_{0,1}, b_{1,0}, b_{1,1})$ and the Receiver (c_0, c_1) respectively.

Theorem 25. *(in OT^1_4-base)* **shows** $protocol_{OT^1_4}(B, C) = funct_{OT^1_4}(B, C)$

Proof. We prove correctness by considering cases on $C = (c_0, c_1)$ and using the correctness of the underlying OT^1_2 protocol assumed in the locale OT^1_4-*base*. □

Next we prove security for each party in turn.

PARTY 1 SECURITY To prove security for the Sender we prove a reduction to the security of the Sender in the underlying OT^1_2. Protocol 2 calls the OT^1_2 protocol three times, thus we bound the advantage of the Sender by $3 \cdot P_1 adv_{OT^1_2}$. The real and simulated views are shown below. Note the difference between the views is the that one calls the real view and one the simulated view of the underlying OT^1_2.

$$\begin{array}{ll}
R_{1,OT^1_4}(B,(c_0,c_1)) = do \, \{ & S_{1,OT^1_4}(B,\_) = do \, \{ \\
\quad S_0 \leftarrow coin; & \quad S_0 \leftarrow coin; \\
\quad S_1 \leftarrow coin; & \quad S_1 \leftarrow coin; \\
\quad S_2 \leftarrow coin; & \quad S_2 \leftarrow coin; \\
\quad S_3 \leftarrow coin; & \quad S_3 \leftarrow coin; \\
\quad S_4 \leftarrow coin; & \quad S_4 \leftarrow coin; \\
\quad S_5 \leftarrow coin; & \quad S_5 \leftarrow coin; \\
\quad a \leftarrow R_{1,OT^1_2}((S_0,S_1),c_0); & \quad a \leftarrow S_{1,OT^1_2}((S_0,S_1),\_); \\
\quad b \leftarrow R_{1,OT^1_2}((S_2,S_3),c_1); & \quad b \leftarrow S_{1,OT^1_2}((S_2,S_3),\_); \\
\quad c \leftarrow R_{1,OT^1_2}((S_4,S_5),c_1); & \quad c \leftarrow S_{1,OT^1_2}((S_2,S_3),\_); \\
\quad return(B,(S_0,S_1,S_2,S_3,S_4,S_5), & \quad return(B,(S_0,S_1,S_2,S_3,S_4,S_5), \\
\qquad\qquad a,b,c)\} & \qquad\qquad a,b,c)\}
\end{array} \tag{7.47}$$

Theorem 26. *(in OT^1_4-base)* **shows** $adv\text{-}P_{1,OT^1_4}(B,C,D) \leq 3 \cdot P_1adv_{OT^1_2}$

Proof. A paper proof would likely state that the reduction holds because Protocol 2 uses three calls to the OT^1_2 protocol. We must work harder. We prove a distinguisher cannot distinguish between the real and simulated views for party 1 in the Protocol 2 with greater advantage than $3 \cdot P_1adv_{OT^1_2}$, by formalising what is commonly called the *hybrid method*. Here we informally describe our proof method.

The main difference between the real and simulated view is that the real view calls R_{1,OT^1_2} three times whereas the simulated view calls S_{1,OT^1_2} three times. To show these two are indistinguishable we define two intermediate views ($inter_{view_i}$ for $i \in \{1,2\}$ that step-wise transform the real view into the simulated view.

$$\begin{array}{ll}
inter_{view_1}(B,(c_0,c_1)) = do \, \{ & inter_{view_2}(B,\_) = do \, \{ \\
\quad S_0 \leftarrow coin; & \quad S_0 \leftarrow coin; \\
\quad S_1 \leftarrow coin; & \quad S_1 \leftarrow coin; \\
\quad S_2 \leftarrow coin; & \quad S_2 \leftarrow coin; \\
\quad S_3 \leftarrow coin; & \quad S_3 \leftarrow coin; \\
\quad S_4 \leftarrow coin; & \quad S_4 \leftarrow coin; \\
\quad S_5 \leftarrow coin; & \quad S_5 \leftarrow coin; \\
\quad a \leftarrow S_{1,OT^1_2}((S_0,S_1),\_); & \quad a \leftarrow S_{1,OT^1_2}((S_0,S_1),\_); \\
\quad b \leftarrow R_{1,OT^1_2}((S_2,S_3),c_1); & \quad b \leftarrow S_{1,OT^1_2}((S_2,S_3),\_); \\
\quad c \leftarrow R_{1,OT^1_2}((S_4,S_5),c_1); & \quad c \leftarrow R_{1,OT^1_2}((S_4,S_5),c_1); \\
\quad return(B,(S_0,S_1,S_2,S_3,S_4,S_5), & \quad return(B,(S_0,S_1,S_2,S_3,S_4,S_5), \\
\qquad\qquad a,b,c)\} & \qquad\qquad a,b,c)\}
\end{array} \tag{7.48}$$

The first intermediate view changes the first call of R_{1,OT^1_2} in the real view to S_{1,OT^1_2}, the second further changes the second call of R_{1,OT^1_2} to S_{1,OT^1_2}. We informally depict this in the diagram below:

$$R_{1,OT^1_2} \prec_{P_1adv_{OT^1_2}} inter_{view_1} \prec_{P_1adv_{OT^1_2}} inter_{view_2} \prec_{P_1adv_{OT^1_2}} S_{1,OT^1_2}$$

Where $A \prec_P B$ denotes that we show a distinguisher has a probability less than P of distinguishing the the probabilistic programs A and B. Once we have proved all three parts in turn we can combine them to show the overall probability a distinguisher has is less than $3 \cdot P_1adv_{OT^1_2}$. In this case Theorem 26 becomes

$$R_{1,OT^1_2} \prec_{3 \cdot P_1adv_{OT^1_2}} S_{1,OT^1_2}.$$

□

PARTY 2 SECURITY To prove security for party 2 we directly use the perfect security result we assume for party 2 in the OT^1_2. The real and simulated views are shown below.

$$\begin{array}{ll}
R_{1,OT^1_4}(B,(c_0,c_1)) = do\ \{ & S_{1,OT^1_4}(B,out_2) = do\ \{ \\
\quad S_0 \leftarrow coin; & \quad S_0 \leftarrow coin; \\
\quad S_1 \leftarrow coin; & \quad S_1 \leftarrow coin; \\
\quad S_2 \leftarrow coin; & \quad S_2 \leftarrow coin; \\
\quad S_3 \leftarrow coin; & \quad S_3 \leftarrow coin; \\
\quad S_4 \leftarrow coin; & \quad S_4 \leftarrow coin; \\
\quad S_5 \leftarrow coin; & \quad S_5 \leftarrow coin; \\
\quad let\ a_0 = S_0 \oplus S_2 \oplus m_{0,0}; & \quad let\ a_0 = (if\ \neg c_0 \wedge \neg c_1\ then\ (S_0 \oplus S_2 \oplus out_2)\ else\ a_0); \\
\quad let\ a_1 = S_0 \oplus S_3 \oplus m_{0,1}; & \quad let\ a_1 = (if\ \neg c_0 \wedge c_1\ then\ (S_0 \oplus S_3 \oplus out_2)\ else\ a_1); \\
\quad let\ a_2 = S_1 \oplus S_4 \oplus m_{1,0}; & \quad let\ a_2 = (if\ c_0 \wedge \neg c_1\ then\ (S_1 \oplus S_4 \oplus out_2)\ else\ a_2); \\
\quad let\ a_3 = S_1 \oplus S_5 \oplus m_{1,1}; & \quad let\ a_3 = (if\ c_0 \wedge c_1\ then\ (S_1 \oplus S_5 \oplus out_2)\ else\ a_3); \\
\quad a \leftarrow R_{2,OT^1_2}((S_0,S_1),c_0); & \quad a \leftarrow S_{2,OT^1_2}((S_0,S_1),\_); \\
\quad b \leftarrow R_{2,OT^1_2}((S_2,S_3),c_1); & \quad b \leftarrow S_{2,OT^1_2}((S_2,S_3),\_); \\
\quad c \leftarrow R_{2,OT^1_2}((S_4,S_5),c_1); & \quad c \leftarrow S_{2,OT^1_2}((S_2,S_3),\_); \\
\quad return(B,(S_0,S_1,S_2,S_3,S_4,S_5), & \quad return(B,(S_0,S_1,S_2,S_3,S_4,S_5), \\
\qquad\qquad a,b,c)\} & \qquad\qquad a,b,c)\}
\end{array} \tag{7.49}$$

Theorem 27. *(in OT^1_4-base)* **shows** *perfect-sec-$P_{2,OT^1_4}(B,C)$*

Proof. To show perfect security we construct a simulator S_{2,OT^1_4} such that it is equal to the real view of the Receiver in Protocol 2. The basis of security for the Receiver is that the assumed perfect security for the Receiver of the underlying OT^1_2 protocol. It was semi-technical to use the assumed result on OT^1_2 as we require the second input to the simulator to be from the functionality for the assumption to be valid (this can be seen after unfolding the definition of *perfect-sec-P_{2,OT^1_2}*) — the challenge is that this input is embedded within the probabilistic program. □

Together Theorems 26 and 27 have shown security for Protocol 2 with respect to the assumptions on the locale *OT^1_4-base*. That is we have proven the security of a protocol that realises OT^1_4 under the assumption that it uses a secure OT^1_2.

7.4.2 *The GMW protocol*

To realise secure circuit evaluation the GMW protocol provides protocols for the secure computation of AND and XOR gates. We formalise these security results. We do so under the assumption of security of a protocol that realises OT^1_4. We make these assumptions in the locale, *gmw-base*, which we show later after we have informally introduced the GMW protocol. The assumptions on the security of the OT^1_4 are exactly the results we proved on OT^1_4 in section 7.4.1.

7.4.2.1 *Securely computing AND and XOR gates*

We show how to securely compute an XOR and AND gate using the GMW protocol. Assume party 1 has input x and party 2 has input y, after sharing and sending the other party the appropriate share party 1 holds the shares (a_1, a_2), and party 2 holds the shares (b_1, b_2) — that is $x = a_1 \oplus b_1$ and $y = a_2 \oplus b_2$.

The GMW protocol provides sub protocols to compute XOR and AND gates on the shared inputs (that have already been shared between the parties).

XOR AND AND PROTOCOLS The functionality for the XOR protocol we wish to compute is as follows,

$$f_{XOR}((a_1, a_2), (b_1, b_2)) = ((a_1 \oplus a_2, b_1 \oplus b_2)) \tag{7.50}$$

The protocol for an XOR gate is given in Protocol 3, all computation is done locally by each party, meaning there is no need for communication between the parties.

Protocol 3. *[XOR gate] To compute an XOR gate the parties can compute the XOR of their shares separately, that is party 1 evaluates $a_1 \oplus a_2$ and party 2 evaluates $b_1 \oplus b_2$.*

Because there is no communication between the parties security is trivial. Correctness comes from the commutativity of the XOR operation.

Securely computing an AND gate is more involved. The functionality we want to evaluate is

$$f_{AND}((a_1, a_2), (b_1, b_2)) = (\sigma, \sigma \oplus (a_1 \oplus b_1) \wedge (a_2 \oplus b_2)) \tag{7.51}$$

where σ is uniformly sampled from $\{0, 1\}$. The sampling of σ ensures neither party alone learns the output, instead they each learn a share, which they must reconstruct together to learn the output. Recall $x = a_1 \oplus b_1$ and $y = a_2 \oplus b_2$ then one can see party 2 learns the $\sigma \oplus (x \wedge y)$ which when xor-ed (reconstructed) with party 1's output, σ, gives the desired result.

A GMW protocol that realises this functionality uses OT^1_4 and is given in Protocol 4 below.

Protocol 4. *[AND gate]*

1. *party 1 samples $\sigma \leftarrow \{0, 1\}$ and constructs s_i as follows:*

b_1	b_2	$(a_1 \oplus b_1) \wedge (a_2 \oplus b_2)$	s_i
0	0	α_0	$s_0 = \sigma \oplus \alpha_0$
0	1	α_1	$s_1 = \sigma \oplus \alpha_1$
1	0	α_2	$s_2 = \sigma \oplus \alpha_2$
1	1	α_3	$s_3 = \sigma \oplus \alpha_3$

2. *The parties compute an OT^1_4 with input (s_0, s_1, s_2, s_3) for party 1 and (b_1, b_2) for party 2.*

3. *party 2 keeps its output of the OT^1_4 while party 1 keeps σ.*

The protocol is correct as both parties hold a share of the output such that when xored together give the desired result. Intuitively, security comes from party 1 constructing all possible results of the computation (in Step 1) and masking it with the random sample σ. The parties then use OT^1_4 to transfer over one and only one of the possible values, the value that party 2 can *decrypt* to form their share — thus the security reduces to the security of the underlying OT^1_4 protocol.

7.4.3 *Formalising Secret Sharing*

Secret sharing schemes [69] are at the core of MPC protocols. They allow inputs to be shared among a number of parties such that the share can only be reconstructed when a critical threshold number of parties combine their shares. We consider secret sharing for two parties, thus both shares are always needed to reconstruct correctly. To formalise such schemes we provide two constants *share* and *reconstruct* that define the sharing scheme. We give their types below.

$$share \;::\; {}'a \Rightarrow ({}'share \times {}'share)\ spmf \tag{7.52}$$

$$reconstruct \;::\; ({}'share \times {}'share) \Rightarrow {}'a\ spmf \tag{7.53}$$

The basic correctness property of a sharing scheme requires that reconstructing a shared input returns the original input.

Definition 28 (Correctness on secret sharing)**.**

$$correct_{share}(input) = (share(input) \rhd (\lambda(s_1, s_2).\ reconstruct(s_1, s_2)) = return(input))$$

Finally we also fix a constant *evaluate* which a set containing all the functions we wish to realise (in the GMW protocol these are AND and XOR).

$$evaluate \;::\; ({}'a \Rightarrow {}'a \Rightarrow {}'a\ spmf)\ set \tag{7.54}$$

In section 7.4.4.1 we show how we instantiate this parameter and show how we prove that all functions in the set are correct with respect our sharing scheme (Lemmas 37 and 38).

7.4.4 *Secret sharing for the GMW*

In the GMW protocol the input from each party to a gate is a bit, thus the parties need to share their input bit between them.

To share a bit x a party flips a coin to obtain a bit, a. The bit a is kept by the party and $x \oplus a$ is sent to the other party; this is often called xor-sharing. To reconstruct the two parties compute the xor of their shares. The formal definitions for this are given below

$$\begin{aligned} &\mathit{share}_{\text{GMW}}(x) = \mathit{do}\ \{\\ &\quad a \leftarrow \mathit{coin};\\ &\quad \mathit{return}(a, x \oplus a)\} \end{aligned} \tag{7.55}$$

$$\mathit{reconstruct}_{\text{GMW}}(a, b) = \mathit{return}(a \oplus b) \tag{7.56}$$

To show correctness of the sharing scheme we show that reconstructing a shared input results in the original input.

Theorem 28. *(in gmw-base)* **shows** $\mathit{correct}_{\mathit{share}_{GMW}}(x)$

7.4.4.1 *Formalising the GMW protocol*

We first consider the correctness of Protocols 3 and 4, which make up the GMW protocol, with respect the the secret sharing scheme we use. Then we show how we prove security for Protocols 3 and 4. In our formalisation here we assume the results we proved on OT_4^1. We make these assumptions in the locale *gmw-base*. We make some type synonyms to make the types in the locale easier to read.

$$\textbf{type-synonym}\ OT_4^1\text{-}\mathit{msgs} = (\mathit{bool} \times \mathit{bool} \times \mathit{bool} \times \mathit{bool}) \tag{7.57}$$

$$\textbf{type-synonym}\ OT_4^1\text{-}\mathit{choice} = (\mathit{bool} \times \mathit{bool}) \tag{7.58}$$

$$\begin{aligned} &\textbf{locale}\ \mathit{gmw\text{-}base} =\\ &\quad \textbf{fixes}\ \mathit{protocol}_{OT_4^1} :: OT_4^1\text{-}\mathit{msgs} \Rightarrow OT_4^1\text{-}\mathit{choice} \Rightarrow (\mathit{unit} \times \mathit{bool})\ \mathit{spmf}\\ &\quad\quad \textbf{and}\ R_{1,OT_4^1} :: OT_4^1\text{-}\mathit{msgs} \Rightarrow OT_4^1\text{-}\mathit{choice} \Rightarrow \text{'}\mathit{view}_{1,OT_4^1}\ \mathit{spmf}\\ &\quad\quad \textbf{and}\ S_{1,OT_4^1} :: OT_4^1\text{-}\mathit{msgs} \Rightarrow \mathit{unit} \Rightarrow \text{'}\mathit{view}_{1,OT_4^1}\ \mathit{spmf}\\ &\quad\quad \textbf{and}\ R_{2,OT_4^1} :: OT_4^1\text{-}\mathit{msgs} \Rightarrow OT_4^1\text{-}\mathit{choice} \Rightarrow \text{'}\mathit{view}_{2,OT_4^1}\ \mathit{spmf}\\ &\quad\quad \textbf{and}\ S_{2,OT_4^1} :: OT_4^1\text{-}\mathit{choice} \Rightarrow \mathit{bool} \Rightarrow \text{'}\mathit{view}_{2,OT_4^1}\ \mathit{spmf}\\ &\quad\quad \textbf{and}\ P_1\mathit{adv}_{OT_4^1} :: \mathit{real}\\ &\quad \textbf{assumes}\ \mathit{protocol}_{OT_4^1}((m_0, m_1, m_2, m_3), \sigma) = \mathit{funct}_{OT_4^1}((m_0, m_1, m_2, m_3), \sigma)\\ &\quad\quad \textbf{and}\ \mathit{adv}\text{-}P_{1,OT_4^1}((m_0, m_1, m_2, m_3), \sigma, D) \leq P_1\mathit{adv}_{OT_4^1}\\ &\quad\quad \textbf{and}\ P_1\mathit{adv}_{OT_4^1} > 0\\ &\quad\quad \textbf{and}\ \mathit{perfect\text{-}sec}\text{-}P_{2,OT_4^1}((m_0, m_1, m_2, m_3), \sigma) \end{aligned} \tag{7.59}$$

Correctness of secret sharing We emphasis here that we are not considering correctness in the sense we consider correctness for deterministic protocols. Instead here we would like to show that the XOR and AND protocols are correct with respect the secret sharing scheme. That is that sharing inputs, computing the protocol and reconstructing gives the desired output of the gate. We showed the functionalities for the XOR and AND gates in Equations 7.50 and 7.51. They are formally encoded in Isabelle as follows.

$$\begin{array}{ll} \mathit{funct}_{\text{AND}}((a_1, a_2), (b_1, b_2)) = \mathit{do}\ \{ & \mathit{funct}_{\text{XOR}}((a_1, a_2), (b_1, b_2)) = \\ \quad \sigma \leftarrow \mathit{coin}; & \quad \mathit{return}(a_1 \oplus a_2, b_1 \oplus b_2) \\ \quad \mathit{return}(\sigma, \sigma \oplus (a_1 \oplus b_1) \wedge (a_2 \oplus b_2))\} & \end{array} \tag{7.60}$$

The probabilistic programs we define that describe Protocols 3 and 4 are given below in Equations 7.61 and 7.62. These protocols take as inputs the already shares held by the parties. Consequently it does not make sense to consider them with respect to the definition of correctness for two party MPC we formalised in section 7.2. Instead we prove more robust statements with respect the secret sharing scheme after instantiating the set *evaluate*.

$$\mathit{protocol}_{\text{XOR}}((a_1, a_2), (b_1, b_2)) = \mathit{return}(a_1 \oplus a_2, b_1 \oplus b_2) \tag{7.61}$$

$$\begin{array}{l} \mathit{protocol}_{\text{AND}}((a_1, a_2), (b_1, b_2)) = \mathit{do}\ \{ \\ \quad \sigma \leftarrow \mathit{coin}; \\ \quad \mathit{let}\ s_0 = \sigma \oplus (a_1 \oplus \mathit{False}) \wedge (b_1 \oplus \mathit{False}); \\ \quad \mathit{let}\ s_1 = \sigma \oplus (a_1 \oplus \mathit{False}) \wedge (b_1 \oplus \mathit{True}); \\ \quad \mathit{let}\ s_2 = \sigma \oplus (a_1 \oplus \mathit{True}) \wedge (b_1 \oplus \mathit{False}); \\ \quad \mathit{let}\ s_3 = \sigma \oplus (a_1 \oplus \mathit{True}) \wedge (b_1 \oplus \mathit{True}); \\ \quad (\_, s) \leftarrow \mathit{protocol}_{OT^1_4}((s_0, s_1, s_2, s_3), (a_2, b_2); \\ \quad \mathit{return}(\sigma, s)\} \end{array} \tag{7.62}$$

We define $\mathit{evaluate} = \{\mathit{evaluate}_{\text{XOR}}, \mathit{evaluate}_{\text{AND}}\}$ as the set of functions we wish to consider — these are defined over the original inputs, and not the shared inputs.

$$\mathit{evaluate}_{\text{AND}}(x, y) = \mathit{return}(x \wedge y) \tag{7.63}$$

$$\mathit{evaluate}_{\text{XOR}}(x, y) = \mathit{return}(x \oplus y) \tag{7.64}$$

We prove Protocol 3 and 4 are correct with respect to the secret sharing. That is sharing the inputs, executing the protocol and reconstructing gives the desired evaluated result. These statements are given in Lemmas 37 and 38.

Lemma 37. *(in gmw-base)*
shows $(share_{\scriptscriptstyle GMW}(x) \rhd (\lambda(s_1, s_2).\ share_{\scriptscriptstyle GMW}(y) \rhd$
$(\lambda(s_3, s_4).\ protocol_{\scriptscriptstyle AND}((s_1, s_3), (s_2, s_4)) \rhd$
$(\lambda(S_1, S_2).\ reconstruct_{\scriptscriptstyle GMW}(S_1, S_2)))) = evaluate_{\scriptscriptstyle AND}(x, y))$

Lemma 38. *(in gmw-base)*
shows $(share_{\scriptscriptstyle GMW}(x) \rhd (\lambda(s_1, s_2).\ share_{\scriptscriptstyle GMW}(y) \rhd$
$(\lambda(s_3, s_4).\ protocol_{\scriptscriptstyle XOR}((s_1, s_3), (s_2, s_4)) \rhd$
$(\lambda(S_1, S_2).\ reconstruct_{\scriptscriptstyle GMW}(S_1, S_2)))) = evaluate_{\scriptscriptstyle XOR}(x, y))$

SECURITY For Protocol 3 that realises the XOR functionality there is no communication between the parties, thus security is trivial. The trivial simulator and proof of security is given in our formalisation **[our_afp]**. Here we focus on the AND gates' security. Note that we are dealing with a non-deterministic functionality so must consider the security definitions given in section 7.2.2. We note we provide the unfolded ideal view in the definitions below, recall in the definitions we make in the non deterministic setting we define the simulator and output separately and combine them into the ideal view.

For Protocol 4 we show a reduction to the security of the OT^1_4 for party 1 and show perfect security for party 2. First we consider party 1. The real view and unfolded ideal view for party 1 are shown below.

$$\begin{aligned}
&R_{\scriptscriptstyle 1,AND}((a_1, a_2), (b_1, b_2)) = do\ \{\\
&\quad \sigma \leftarrow coin;\\
&\quad let\ s_0 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus False);\\
&\quad let\ s_1 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus True);\\
&\quad let\ s_2 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus False);\\
&\quad let\ s_3 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus True);\\
&\quad V \leftarrow R_{1,OT^1_4}((s_0, s_1, s_2, s_3), (b_1, b_2));\\
&\quad (\_, s) \leftarrow protocol_{OT^1_4}((s_0, s_1, s_2, s_3), (b_1, b_2);\\
&\quad return(((a_1, a_2), \sigma, V), (\sigma, s))\}
\end{aligned} \tag{7.65}$$

$$\begin{aligned}
&ideal\text{-}view_{\scriptscriptstyle 1,AND}((a_1, a_2), (b_1, b_2), \sigma) = do\ \{\\
&\quad let\ s_0 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus False);\\
&\quad let\ s_1 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus True);\\
&\quad let\ s_2 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus False);\\
&\quad let\ s_3 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus True);\\
&\quad V \leftarrow S_{1,OT^1_4}((s_0, s_1, s_2, s_3), ());\\
&\quad return(((a_1, a_2), \sigma, V), (\sigma, \sigma \oplus ((a_1 \oplus b_1) \wedge (a_2 \oplus b_2))))\}
\end{aligned} \tag{7.66}$$

As in the proof of the OT^1_4 construction the difference between the views is whether the simulator or real view of the underlying OT^1_4 is called. Thus the proof of security for party 1 follows a similar argument, the statement of security is given below.

Theorem 29. *(in gmw-base)*
shows $adv\text{-}P_{1,AND}((a_1, a_2), (b_1, b_2), D) \leq P_1 adv_{OT^1_4}$

Proof. The proof is similar in construction to the reduction we show for party 1 of the OT^1_4 protocol in Theorem 26. The only difference is that we only need to show one reduction, as opposed to the three we showed there. □

For party 2 we show perfect security, like in the case of the Receiver in the proof of OT^1_4 from OT^1_2. The views and the statement of security are shown below.

$$\begin{aligned}
&R_{2,AND}((a_1, a_2), (b_1, b_2)) = do\ \{\\
&\quad \sigma \leftarrow coin;\\
&\quad let\ s_0 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus False);\\
&\quad let\ s_1 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus True);\\
&\quad let\ s_2 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus False);\\
&\quad let\ s_3 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus True);\\
&\quad V \leftarrow R_{2,OT^1_4}((s_0, s_1, s_2, s_3), (b_1, b_2));\\
&\quad (\_, out_2) \leftarrow protocol_{OT^1_4}((s_0, s_1, s_2, s_3), (b_1, b_2);\\
&\quad return(((b_1, b_2), V), (\sigma, out_2))\}
\end{aligned} \tag{7.67}$$

$$\begin{aligned}
&ideal\text{-}view_{2,AND}((b_1, b_2), (a_1, a_2), out_2) = do\ \{\\
&\quad V \leftarrow S_{2,OT^1_4}((b_0, b_1), out_2);\\
&\quad let\ s_1 = out_2 \oplus (a_1 \oplus b_1) \wedge (a_2 \oplus b_2);\\
&\quad return(((b_1, b_2), V), (s_1, out_2))\}
\end{aligned} \tag{7.68}$$

Theorem 30. *(in gmw-base)*
shows $perfect\text{-}sec\text{-}P_{2,AND}((a_1, a_2), (b_1, b_2))$

Proof. We use the assumption that the underlying OT^1_4 is perfectly secure for party 2 to replace R_{2,OT^1_4} for S_{2,OT^1_4} in the real view which brings it closer to equaling the simulator. Then we consider the cases on the inputs a_1, a_2, b_1, b_2 to dismiss the proof. □

Theorems 30 and 29 show security in the semi-honest model for Protocol 4.

7.5 SECURE MULTIPLICATION PROTOCOL

We now present a protocol that computes a secure multiplication in $\mathbb{Z}_q$. The functionality we consider is given in Equation 7.69 below,

$$f_{mult}(x, y) = (s, (x \cdot y - s)\ mod\ q) \tag{7.69}$$

where $s \leftarrow \mathbb{Z}_q$. In the functionality we uniformly sample s so that neither output reveals the result of the multiplication on its own. Each party receives a share of the result. Clearly, to reconstruct the result both outputs must added together.

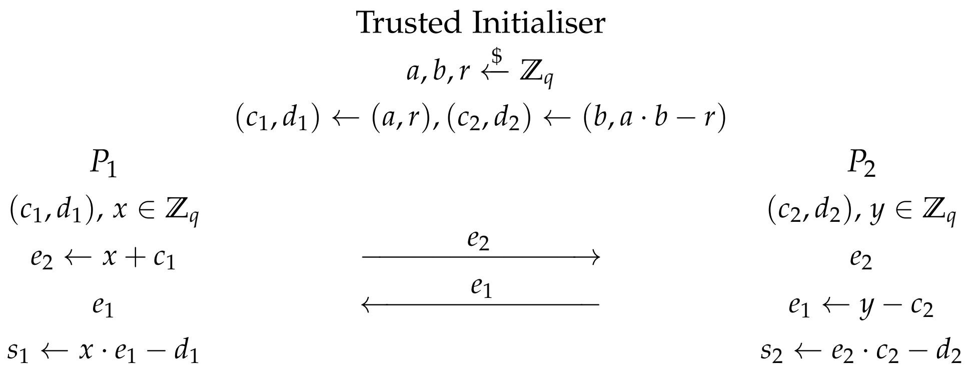

Trusted Initialiser
$a, b, r \xleftarrow{\$} \mathbb{Z}_q$
$(c_1, d_1) \leftarrow (a, r), (c_2, d_2) \leftarrow (b, a \cdot b - r)$
P_1
$(c_1, d_1), x \in \mathbb{Z}_q$
$e_2 \leftarrow x + c_1$
e_1
$s_1 \leftarrow x \cdot e_1 - d_1$
e_2
e_1
P_2
$(c_2, d_2), y \in \mathbb{Z}_q$
e_2
$e_1 \leftarrow y - c_2$
$s_2 \leftarrow e_2 \cdot c_2 - d_2$

Figure 7.3: A protocol for secure multiplication

The protocol provides perfect security for both parties. To achieve this the protocol requires some pre-generation of elements to be distributed to the parties. This is known in MPC as the preprocessing model [9], where the parties run an offline phase to generate correlated random triples — sometimes called Beaver triples — that are used to perform fast secure multiplications in an online phase. For this task we assume a trusted initialiser exists. The preprocessing model is widely accepted in the cryptographic community as it allows for stronger notions of security (generally perfect security guarantees) whilst the trusted initialiser does not learn either party's input. The protocol that realises the functionality given in Equation 7.69 is given in Figure 7.3. In the protocol all operations are computed in $\mathbb{Z}_q$.

Intuitively, security results from the messages being sent in the protocol always being masked by some randomness. There are only two messages sent in the protocol. In the message party one sends, e_2, the input (x) is masked by the uniform sample, c_1. Likewise in the message party two sends, e_1, the input (y) is masked by the uniform sample, c_2. Both c_1 and c_2 are unique samples not used elsewhere in the protocol so the masking provides a one time pad, so perfect security holds for both parties.

7.5.1 *Formalising the protocol*

Like the AND gate GMW protocol we considered in the previous section the protocol we consider here is also non-deterministic, thus we use the definitions of security from section 7.2.2. In particular we do not need to show correctness as this is captured in the extended views considered in the security statement. We construct a locale in which to consider the protocol. In the locale we fix the size of the field q and assume it is greater than 0.

$$\begin{aligned}
&\textbf{locale}\ \textit{secure-mult} = \\
&\quad \textbf{fixes}\ q\ ::\ \textit{nat} \\
&\quad \textbf{assumes}\ q > 0
\end{aligned} \tag{7.70}$$

The functionality we consider in this protocol is encoded in Isabelle as follows.

$$\begin{aligned} &\mathit{funct}_{mult}(x,y) = do\ \{ \\ &\quad s \leftarrow \mathit{samp\text{-}uniform}(q); \\ &\quad \mathit{return}(s, (x \cdot y - s)\ \mathit{mod}\ q)\} \end{aligned} \tag{7.71}$$

We note as we are dealing with naturals we must be careful for the subtraction not to bottom out. To account for this in the formalisation we actually return $(x \cdot y + (q - s))$ $\mathit{mod}\ q$ as the second output, however do not write this here to avoid clutter.

To formalise the protocol we first define the output of the trusted initialiser.

$$\begin{aligned} &TI = do\ \{ \\ &\quad a \leftarrow \mathit{samp\text{-}uniform}(q); \\ &\quad b \leftarrow \mathit{samp\text{-}uniform}(q); \\ &\quad r \leftarrow \mathit{samp\text{-}uniform}(q); \\ &\quad \mathit{return}((a,r),(b, a \cdot b - r)\ \mathit{mod}\ q)\} \end{aligned} \tag{7.72}$$

We have perfect security for both parties. For party 1 the real view and unfolded ideal view are as follows.

$$\begin{aligned} &R_1(x,y) = do\ \{ \\ &\quad ((c_1,d_1),(c_2,d_2)) \leftarrow TI; \\ &\quad let\ e_1 = (y - c_2)\ mod\ q; \\ &\quad let\ e_2 = (x + c_1)\ mod\ q; \\ &\quad let\ s_1 = (x \cdot e_1 - d_1)\ mod\ q; \\ &\quad let\ s_2 = (e_2 \cdot c_2 - d_2)\ mod\ q; \\ &\quad \mathit{return}(x, c_1, d_1, e_1, s_1, s_2)\} \end{aligned} \tag{7.73}$$

$$\begin{aligned} &\mathit{ideal\text{-}view}_{1,mult}(x,y,s_1) = do\ \{ \\ &\quad c_1 \leftarrow \mathit{samp\text{-}uniform}(q); \\ &\quad e_1 \leftarrow \mathit{samp\text{-}uniform}(q); \\ &\quad let\ d_1 = (x \cdot e_1 - s_1)\ mod\ q; \\ &\quad let\ s_2 = (x \cdot y - s_1)\ mod\ q; \\ &\quad \mathit{return}(x, c_1, d_1, e_1, s_1, s_2)\} \end{aligned} \tag{7.74}$$

To show perfect security we prove that the two views given in 7.73 and are equal when s_1, the input to the simulator, is the first output of the functionality.

Theorem 31. *(in secure-mult)* **shows** *perfect-sec-*$P_{1,mult}(x,y)$

Proof. The proof involves a series of small equality steps between intermediate probabilistic programs. In particular, in the series of intermediate programs we manipulate the real and ideal views. In the ideal view the samples c_1, e_1 and s_1 are uniformly random and independent from each other, x and y. We show $s_2 = (x \cdot y - s_1)\ mod\ q$ and $d_1 = (x \cdot e_1 - s_1)\ mod\ q$. By showing these relationships, and only these relationships, hold for the real view too we show the two views are equal. Much of the proof effort required in Isabelle was to show various equalities hold concerning arithmetic modulo q. The built in techniques to aid with such arithmetic were not able to deal with the results required here easily. □

For party 2 the real and unfolded ideal view are as follows.

$$\begin{aligned}
&R_2(x,y) = do\ \{\\
&\quad ((c_1,d_1),(c_2,d_2)) \leftarrow TI;\\
&\quad let\ e_1 = (y - c_2)\ mod\ q;\\
&\quad let\ e_2 = (x + c_1)\ mod\ q;\\
&\quad let\ s_1 = (x \cdot e_1 - d_1)\ mod\ q;\\
&\quad let\ s_2 = (e_2 \cdot c_2 - d_2)\ mod\ q;\\
&\quad return(y, c_2, d_2, e_2, s_1, s_2)\}
\end{aligned} \tag{7.75}$$

$$\begin{aligned}
&ideal\text{-}view_{2,mult}(y,x,s_2) = do\ \{\\
&\quad c_2 \leftarrow samp\text{-}uniform(q);\\
&\quad e_2 \leftarrow samp\text{-}uniform(q);\\
&\quad let\ d_2 = (e_2 \cdot c_2 - s_2)\ mod\ q;\\
&\quad let\ s_2 = (x \cdot y - s_2)\ mod\ q;\\
&\quad return((y, c_2, d_2, e_2), s_1, s_2)\}
\end{aligned} \tag{7.76}$$

Again, like for party 1, we show perfect security.

Theorem 32. *(in secure-mult)* **shows** *perfect-sec-*$P_{2,mult}(x,y)$

Proof. The proof here is similar to the proof of Theorem 32. The relationships to consider are $s_2 = (x \cdot y - s_1)\ mod\ q$ and $d_2 = (e_2 \cdot c_2 - s_2)\ mod\ q$. □

Together, Theorems 31 and 32 show security in the semi-honest model for the Protocol given in Figure 7.3.

In this chapter we have shown how we formalised semi-honest security for two party protocols. We have provided multiple instances of protocols that realise these definitions: starting from OT^1_2 building to the GMW protocol and a secure multiplication protocol. Moreover, in our proof of the GMW protocol we have shown how Isabelle's module system can be used to modularise the proof, for example by assuming the security properties on OT^1_4 when considering the GMW protocol.

8 MALICIOUS SECURITY

8.1 INTRODUCTION

In this chapter we consider the malicious security model. In the malicious model the adversary is allowed to totally corrupt the party (or parties) it controls. As we work in the two party setting we only need to consider the case where one party is corrupted as we cannot guarantee anything if both parties are corrupted.

CHAPTER OUTLINE In Section 8.2 we formalise the definitions of malicious security we gave in Section 2.4.2. We then instantiate our definitions in Section 8.3 to prove the OT_2^1 protocol from [44, p190] secure.

The work in this chapter has been published in [20, 23].

8.2 FORMALISING THE DEFINITIONS

To make the definitions of malicious security we, as usual, construct a locale (*malicious-base*, given in Equation 8.13) and fix the parameters we require to make our definitions. In this case we fix: the functionality (*funct*), the protocol output (*protocol*), the real view of each party (R_1 and R_2), and the simulators ((S_1^1, S_1^2) and (S_2^1, S_2^2)) — this is the simulator that interacts in the ideal model. The roles of each component of the simulators will become clear when we define their types and the ideal model. The real view of each party is the transcript a party sees when the adversary sends all messages in place of the corrupted party. In contrast, the honest party follows the instructions of the protocol. Before we define the locale, *malicious-base* we construct some type synonyms for readability, in particular we consider the types of the simulators.

We first consider party 1. We define the type of the first adversary that takes part in the ideal game. This adversary has two components, we define their types separately in Equations 8.1 and 8.2 and combine the types in Equation 8.3. The first component of the ideal adversary takes as input the input of party 1 and the auxiliary input and outputs the input it wishes to give to the trusted party on behalf of party 1 as well as some auxiliary output, that can be passed to the second component.

$$\textbf{type-synonym}\ ('in_1, 'aux, 's_1)\ \mathcal{A}_{ideal,1,P_1} = \\ 'in_1 \Rightarrow 'aux \Rightarrow ('in_1 \times 's_1)\ spmf \quad (8.1)$$

The second component of the ideal adversary for party 1 is allowed to also see the output of the protocol, as given to it by the trusted party as well as the auxiliary output (state) outputted by the first component of the adversary. This part of the adversary outputs whatever it likes — we represent this by allowing it to output something of abstract type.

$$\textbf{type-synonym}\ ('in_1,'aux,'out_1,'s_1,'\mathcal{A}_{out_1})\ \mathcal{A}_{ideal,2,P_1} = \\ 'in_1 \Rightarrow 'aux \Rightarrow 'out_1 \Rightarrow 's_1 \Rightarrow '\mathcal{A}_{out_1}\ spmf \tag{8.2}$$

We combine the two components of the adversary for the ideal game as a tuple.

$$\textbf{type-synonym}\ ('in_1,'aux,'out_1,'s_1,'\mathcal{A}_{out_1})\ \mathcal{A}_{ideal,P_1} = \\ ('in_1,'aux,'s_1)\ \mathcal{A}_{ideal,1,P_1} \times \\ ('in_1,'aux,'out_1,'s_1,'\mathcal{A}_{out_1})\ \mathcal{A}_{ideal,2,P_1} \tag{8.3}$$

To prove security there must be simulators that can simulate each component of the ideal game adversaries. Again we consider the types of the components separately in Equations 8.1 and 8.2 before combining them in Equation 8.3. The simulators have the same types as the corresponding components of the ideal game adversaries with the additional input of the real world adversary, that is the adversary that sends the messages on behalf of the corrupted (party 1, in this case) in the real world.

$$\textbf{type-synonym}\ ('\mathcal{A}_{real,P_1},'in_1,'aux,'s_1)\ S_{1,P_1} = \\ '\mathcal{A}_{real,P_1} \Rightarrow 'in_1 \Rightarrow 'aux \Rightarrow ('in \times 's_1)\ spmf \tag{8.4}$$

$$\textbf{type-synonym}\ ('\mathcal{A}_{real,P_1},'in_1,'aux,'out_1,'s_1,'\mathcal{A}_{out_1})\ S_{2,P_1} = \\ \mathcal{A}_{real,P_1} \Rightarrow 'in_1 \Rightarrow 'aux \Rightarrow 'out_1 \Rightarrow 's_1 \Rightarrow '\mathcal{A}_{out_1}\ spmf \tag{8.5}$$

$$\textbf{type-synonym}\ ('\mathcal{A}_{real,P_1},'in_1,'aux,'out_1,'s_1,'\mathcal{A}_{out_1})\ S_{P_1} = \\ ('\mathcal{A}_{real,P_1},'in_1,'aux,'s_1)\ S_{1,P_1} \times \\ ('\mathcal{A}_{real,P_1},'in_1,'aux,'out_1,'s_1,'\mathcal{A}_{out_1})\ S_{2,P_1} \tag{8.6}$$

The type synonym's above have defined the types we need for party 1. Below we make the analogous definitions for party 2.

$$\textbf{type-synonym}\ ('in_2,'aux,'s_2)\ \mathcal{A}_{ideal,1,P_2} = \\ 'in_2 \Rightarrow 'aux \Rightarrow ('in_2 \times 's_2)\ spmf \tag{8.7}$$

$$\textbf{type-synonym}\ ('in_2,'aux,'out_2,'s_2,'\mathcal{A}_{out_2})\ \mathcal{A}_{ideal,2,P_2} = \\ 'in_2 \Rightarrow 'aux \Rightarrow 'out_2 \Rightarrow 'aux_{P_2,S_1} \Rightarrow '\mathcal{A}_{out_2}\ spmf \tag{8.8}$$

$$\textbf{type-synonym}\ ('in_2,'aux,'out_2,'s_2,'\mathcal{A}_{out_2})\ \mathcal{A}_{ideal,P_2} = \\ ('in_2,'aux,'s_2)\ \mathcal{A}_{ideal,1,P_2} \times ('in_2,'aux,'out_2,'s_2,'\mathcal{A}_{out_2})\ \mathcal{A}_{ideal,2,P_2} \tag{8.9}$$

$$\begin{aligned}&\textbf{type-synonym}\ (\text{‘}\mathcal{A}_{\mathit{real},P_2},\text{‘}\mathit{in}_2,\text{‘}\mathit{aux},\text{‘}s_2)\ S_{1,P_2} =\\ &\qquad \text{‘}\mathcal{A}_{\mathit{real},P_2} \Rightarrow \text{‘}\mathit{in}_2 \Rightarrow \text{‘}\mathit{aux} \Rightarrow (\text{‘}\mathit{in}_2 \times \text{‘}s_2)\ \mathit{spmf}\end{aligned} \tag{8.10}$$

$$\begin{aligned}&\textbf{type-synonym}\ (\text{‘}\mathcal{A}_{\mathit{real},P_1},\text{‘}\mathit{in}_2,\text{‘}\mathit{aux},\text{‘}\mathit{out}_2,\text{‘}s_2,\text{‘}\mathcal{A}_{\mathit{out}_2})\ S_{2,P_2} =\\ &\qquad \text{‘}\mathcal{A}_{\mathit{real},P_2} \Rightarrow \text{‘}\mathit{in}_2 \Rightarrow \text{‘}\mathit{aux} \Rightarrow \text{‘}\mathit{out}_2 \Rightarrow \text{‘}s_2 \Rightarrow \text{‘}\mathcal{A}_{\mathit{out}_2}\ \mathit{spmf}\end{aligned} \tag{8.11}$$

$$\begin{aligned}&\textbf{type-synonym}\ (\text{‘}\mathcal{A}_{\mathit{real},P_2},\text{‘}\mathit{in}_2,\text{‘}\mathit{aux},\text{‘}\mathit{out}_2,\text{‘}s_2,\text{‘}\mathcal{A}_{\mathit{out}_2})\ S_{P_2} =\\ &\qquad (\text{‘}\mathcal{A}_{\mathit{real},P_2},\text{‘}\mathit{in}_2,\text{‘}\mathit{aux},\text{‘}s_2)\ S_{1,P_2}\\ &\qquad\qquad \times (\text{‘}\mathcal{A}_{\mathit{real},P_2},\text{‘}\mathit{in}_2,\text{‘}\mathit{aux},\text{‘}\mathit{out}_2,\text{‘}s_2,\text{‘}\mathcal{A}_{\mathit{out}_2})\ S_{2,P_2}\end{aligned} \tag{8.12}$$

Using these type synonyms we construct the locale *malicious-base* as follows.

$$\begin{aligned}&\textbf{locale}\ \mathit{malicious\text{-}base} =\\ &\quad \textbf{fixes}\ \mathit{funct}\ ::\ \text{‘}\mathit{in}_1 \Rightarrow \text{‘}\mathit{in}_2 \Rightarrow (\text{‘}\mathit{out}_1 \times \text{‘}\mathit{out}_2)\ \mathit{spmf}\\ &\quad\ \ \textbf{and}\ \mathit{protocol}\ ::\ \text{‘}\mathit{in}_1 \Rightarrow \text{‘}\mathit{in}_2 \Rightarrow (\text{‘}\mathit{out}_1 \times \text{‘}\mathit{out}_2)\ \mathit{spmf}\\ &\quad\ \ \textbf{and}\ S_1^1\ ::\ (\text{‘}\mathcal{A}_{\mathit{real},P_1},\text{‘}\mathit{in}_1,\text{‘}\mathit{aux},\text{‘}s_1)\ S_{1,P_1}\\ &\quad\ \ \textbf{and}\ S_1^2\ ::\ (\text{‘}\mathcal{A}_{\mathit{real},P_1},\text{‘}\mathit{in}_1,\text{‘}\mathit{aux},\text{‘}\mathit{out}_1,\text{‘}s_1,\text{‘}\mathcal{A}_{\mathit{out}_1})\ S_{2,P_1}\\ &\quad\ \ \textbf{and}\ R_1\ ::\ \text{‘}\mathit{in}_1 \Rightarrow \text{‘}\mathit{in}_2 \Rightarrow \text{‘}\mathit{aux} \Rightarrow \text{‘}\mathcal{A}_{\mathit{real},P_1} \Rightarrow (\text{‘}\mathcal{A}_{\mathit{out}_1} \times \text{‘}\mathit{out}_2)\ \mathit{spmf}\\ &\quad\ \ \textbf{and}\ S_2^1\ ::\ (\text{‘}\mathcal{A}_{\mathit{real},P_2},\text{‘}\mathit{in}_2,\text{‘}\mathit{aux},\text{‘}s_2)\ S_{1,P_2}\\ &\quad\ \ \textbf{and}\ S_2^2\ ::\ (\text{‘}\mathcal{A}_{\mathit{real},P_1},\text{‘}\mathit{in}_2,\text{‘}\mathit{aux},\text{‘}\mathit{out}_2,\text{‘}s_2,\text{‘}\mathcal{A}_{\mathit{out}_2})\ S_{2,P_2}\\ &\quad\ \ \textbf{and}\ R_2\ ::\ \mathit{in}_1 \Rightarrow \text{‘}\mathit{in}_2 \Rightarrow \text{‘}\mathit{aux} \Rightarrow \text{‘}\mathcal{A}_{\mathit{real},P_2} \Rightarrow (\text{‘}\mathit{out}_1 \times \text{‘}\mathcal{A}_{\mathit{out}_2})\ \mathit{spmf}\end{aligned} \tag{8.13}$$

In the same way as the semi-honest setting for a protocol to be correct we require the *funct* and *protocol* to be equal. Unlike in the semi-honest setting correctness and security are not linked. When one party (out of two) is totally corrupt there can be no guarantees that either party obtains the correct output. However, if the protocol is run honestly we still want the correctness property to hold.

Definition 29 (Correctness)**.**

$$\mathit{correct}(x,y) = (\mathit{protocol}(x,y) = \mathit{funct}(x,y))$$

We now make our definitions of malicious security. The ideal games used to define security for each party require a call to the trusted party. This call returns the corresponding output of the functionality, thus we make the following abbreviation.[1]

$$\mathit{trusted\text{-}party}(x,y) = \mathit{funct}(x,y) \tag{8.14}$$

The ideal game for party 1 is defined as follows. This corresponds to the informal definition Definition 11.

[1] Abbreviations in Isabelle are used to rename variables, however in proofs the abbreviations are automatically unfolded. In this instance we use the abbreviation to make it clear what the role of the trusted party is without wanting to make a new definition.

$$\begin{aligned}
&\mathit{ideal\text{-}model}_1(x, y, z, \mathcal{A}) = \mathit{do}\ \{\\
&\quad \mathit{let}\ (\mathcal{A}_1, \mathcal{A}_2) = \mathcal{A};\\
&\quad (x', \mathit{aux}_{out}) \leftarrow \mathcal{A}_1(x, z);\\
&\quad (\mathit{out}_1, \mathit{out}_2) \leftarrow \mathit{trusted\text{-}party}(x', y);\\
&\quad \mathit{out}_1' \leftarrow \mathcal{A}_2(x', z, \mathit{out}_1, \mathit{aux}_{out});\\
&\quad \mathit{return}(\mathit{out}_1', \mathit{out}_2)\}
\end{aligned} \tag{8.15}$$

Using this we define the ideal view for party 1 (Equation 8.16) which allows us to define perfect security and the advantage for party 1. We make these definitions now.

$$\mathit{mal\text{-}ideal}_1(x, y, z, \mathcal{A}) = \mathit{ideal\text{-}model}_1(x, y, z, (S_1^1(\mathcal{A}), S_1^2(\mathcal{A}))) \tag{8.16}$$

Here $\mathcal{A}$ consists of a tuple of algorithms, one for each round of the protocol — that is one algorithm (sub adversary) to send messages on behalf of the corrupted party for each round of the protocol.

As in the semi-honest case we either show perfect security or show the views are indistinguishable — in which case we refer to the advantage that a distinguisher has of distinguishing them. For perfect security we require equality between the views.

Definition 30 (Perfect Security, party 1)**.**

$$\mathit{perfect\text{-}sec\text{-}P}_1(x, y, z, \mathcal{A}) = (R_1(x, y, z, \mathcal{A}) = \mathit{ideal}_1(x, y, z, \mathcal{A}))$$

Definition 31 (Advantage for party 1)**.**

$$\begin{aligned}
&\mathit{adv\text{-}P}_1(x, y, z, \mathcal{A}, D) =\\
&\quad (|\mathcal{P}[(R_1(x, y, z, \mathcal{A}) \rhd D) = \mathit{True}] - \mathcal{P}[(\mathit{mal\text{-}ideal}_1(x, y, z, \mathcal{A}) \rhd D) = \mathit{True}]|)
\end{aligned}$$

We make the analogous definitions for party 2.

$$\begin{aligned}
&\mathit{ideal\text{-}model}_2(x, y, z, \mathcal{A}) = \mathit{do}\ \{\\
&\quad \mathit{let}\ (\mathcal{A}_1, \mathcal{A}_2) = \mathcal{A};\\
&\quad (y', \mathit{aux}_{out}) \leftarrow \mathcal{A}_1(y, z);\\
&\quad (\mathit{out}_1, \mathit{out}_2) \leftarrow \mathit{trusted\text{-}party}(x, y');\\
&\quad \mathit{out}_2' \leftarrow \mathcal{A}_2(y', z, \mathit{out}_2, \mathit{aux}_{out});\\
&\quad \mathit{return}(\mathit{out}_1, \mathit{out}_2')\}
\end{aligned} \tag{8.17}$$

$$\mathit{mal\text{-}ideal}_2(x, y, z, \mathcal{A}) = \mathit{ideal\text{-}model}_2(x, y, z, (S_2^1(\mathcal{A}), S_2^1(\mathcal{A}))) \tag{8.18}$$

Definition 32 (Perfect security for party 2)**.**

$$\mathit{perfect\text{-}sec\text{-}P}_2(x, y, z, \mathcal{A}) \equiv (\mathit{mal\text{-}ideal}_2(x, y, z, \mathcal{A}) = R_2(x, y, z, \mathcal{A}))$$

The advantage of a distinguisher to be is defined as follows.

Definition 33 (Advantage for party 2)**.**

$$\begin{aligned}&\textit{adv-}P_2(x,y,z,\mathcal{A},D) =\\&\quad(|\mathcal{P}[(R_2(x,y,z,\mathcal{A}) \rhd D) = \textit{True}] - \mathcal{P}[(\textit{mal-ideal}_2(x,y,z,\mathcal{A}) \rhd D) = \textit{True}]|)\end{aligned}$$

We make two remarks about our definition of the ideal model (Equations 8.15 and 8.17); the first concerns aborts and the second the state of the adversary.

1. We do not explicitly model the abort procedures (defined informally in Definition 11), this is because they are covered by the type of *spmf*. When the adversary provides output in the probabilistic program it could also output None, this captures the abort process as it terminates the probabilistic program with no output.

2. As we split the adversary into two parts it must be allowed to pass state. This is rarely considered or noted in paper proofs.

The work of Haagh et al. [41] formalises the same malicious model (active security model) in EasyCrypt, however as we discuss in the related work section in the conclusion (section 9.1.1) a meta (paper) theorem was required to link the formalisation to the paper definitions.

8.3 A PROTOCOL REALISING OT^1_2 IN THE MALICIOUS SETTING

In this section we show our definitions for malicious security are satisfied by the OT^1_2 protocol from [44, p.190]. This protocol is considered in the hybrid model as it uses a call to a Zero Knowledge Proof of Knowledge (ZKPoK) functionality for the DH relation (F^{DH}_{ZK}). A Zero Knowledge Proof is an extension of the idea of a Σ-protocol where a Prover is able to convince another party, the Verifier, that they know a value x, without leaking any information to the Verifier except that they know x. This is seen in Step 4 of the protocol.

The protocol uses a cyclic group G (where it is assumed the DDH assumption holds) of order q with generator g and is run as follows:

Protocol 5. *Let party 1 be the sender (S) and party 2 be the receiver (R).*

1. *S has input $(m_0, m_1) \in G^2$ and R has input $\sigma \in \{0,1\}$.*

2. *R uniformly samples $\alpha_0, \alpha_1, r \leftarrow \{1,...,q\}$ and computes $h_0 = g^{\alpha_0}$, $h_1 = g^{\alpha_1}$, $a = g^r$, $b_0 = h_0^r \cdot g^\sigma$ and $b_1 = h_1^r \cdot g^\sigma$.*

3. *S checks $(h_0, h_1, a, b_0, b_1) \in G^5$, otherwise it aborts.*

4. *Let $h = h_0/h_1$ and $b = b_0/b_1$. R proves to S that (h,a,b) is a DH tuple, using ZKPoK. That is R proves the relation*

$$\mathcal{R}_{DH} = \{((h,a,b),r).\ a = g^r \wedge b = h^r\}$$

5. *If S accepts the proof it continues and uniformly samples* $u_0, v_0, u_1, v_1 \leftarrow \{1, ..., q\}$, *and computes* (e_0, e_1) *and sends the tuple to R:*
$e_0 = (w_0, z_0)$ *where* $w_0 = a^{u_0} \cdot g^{v_0}$ *and* $z_0 = b_0^{u_0} \cdot h_0^{v_0} \cdot m_0$.
$e_1 = (w_1, z_1)$ *where* $w_1 = a^{u_1} \cdot g^{v_1}$ *and* $z_1 = (\frac{b_1}{g})^{u_1} \cdot h_1^{v_1} \cdot m_1$.

6. *R outputs* $\frac{z_\sigma}{w_\sigma^{\alpha_\sigma}}$ *and S outputs nothing.*

Here '$\cdot$' denotes multiplication in the group and $\frac{a}{b}$ denotes multiplication by the inverse of b in the group (in the case where $a, b \in G$).

Correctness of the protocol can be seen if one splits the proof into the cases on the value of σ. Intuitively security for the receiver is upheld because σ is sent only as an exponent of the generator which is masked by random group element and the receiver can learn at most one of the sender's messages due to the construction of the DDH tuple, which the sender is satisfied with after the ZKPoK.

8.3.1 *Formally proving* OT_2^1 *secure in the malicious setting*

In this section we will discuss our formalisation of security of Protocol 5.

As usual we construct a locale where we fix the parameters of the protocol, in this case the cyclic group G.

$$\begin{aligned}&\textbf{locale } \textit{mal-ot} =\\ &\quad \textbf{fixes } G :: \textit{'grp cyclic-group}\\ &\quad \textbf{assumes } |G| > 0\\ &\qquad \textbf{and } \textit{prime}(|G|)\end{aligned} \tag{8.19}$$

To define the protocol and views we first define the ideal ZKPoK functionality. In this instance the functionality is encoded into Isabelle as follows.

$$F_{ZK}^{DH}((h, a, b), (((h', a', b'), r)) = \\ \textit{return}(a = g^r \wedge b = h^r \wedge (h, a, b) = (h', a', b'), \_). \tag{8.20}$$

In the context of Protocol 5, party 1 in the functionality is the Sender and party 2 the Receiver. Both parties input a tuple, these could be different as the parties may be malicious. The functionality returns a boolean dependent on whether the relation is true to party 1, and nothing to party 2. In this instance the relation is the Diffie Helman relation. party 1 also learns if the input made by party 2 is the same as their input.

First we prove correctness of the protocol. The parameter $\textit{protocol}_{OT_2^1, mal}$ is defined as follows.

```
$protocol_{OT^1_2,mal}((m_0, m_1), \sigma) = do \{$
  $r \leftarrow samp\text{-}uniform(|G|);$
  $\alpha_0 \leftarrow samp\text{-}uniform(|G|);$
  $\alpha_1 \leftarrow samp\text{-}uniform(|G|);$
  $let\ h_0 = g^{\alpha_0};$
  $let\ h_1 = g^{\alpha_1};$
  $let\ a = g^r;$
  $let\ b_0 = h_0^r \otimes g^{(if\ \sigma\ then\ 1\ else\ 0)};$
  $let\ b_1 = h_1^r \otimes g^{(if\ \sigma\ then\ 1\ else\ 0)};$
  $let\ h = h_0 \otimes invh_1;$
  $let\ b = b_0 \otimes invb_1;$
  $\_ \leftarrow assert(a = g^r \wedge b = h^r);$
  $u_0 \leftarrow samp\text{-}uniform(|G|);$
  $u_1 \leftarrow samp\text{-}uniform(|G|);$
  $v_0 \leftarrow samp\text{-}uniform(|G|);$
  $v_1 \leftarrow samp\text{-}uniform(|G|);$
  $let\ z_0 = b_0^{u_0} \otimes h_0^{v_0} \otimes m_0;$
  $let\ w_0 = a^{u_0} \otimes g^{v_0};$
  $let\ e_0 = (w_0, z_0);$
  $let\ z_1 = (b_1 \otimes inv(g))^{u_1} \otimes h_1^{v_1} \otimes m_1;$
  $let\ w_1 = a^{u_1} \otimes g^{v_1};$
  $let\ e_1 = (w_1, z_1);$
  $return((), if\ \sigma\ then\ (z_1 \otimes inv(w_1^{\alpha_1}))\ else\ (z_0 \otimes inv(w_0^{\alpha_0})))\}$
```

Theorem 33. *(in mal-ot)*
assumes $m_0 \in G$
and $m_1 \in G$
shows $funct_{OT^1_2}((m_0, m_1), \sigma) = protocoll_{OT^1_2,mal}((m_0, m_1), \sigma)$

Proof. Isabelle had to be helped more extensively in the rewriting of terms here compared to other proofs of correctness, mainly Isabelle requires some group identities to be provided as rewrite rules. This was due to the more complex constructions in the protocol. □

To prove security of Protocol 5 we must first formalise a variant of the DDH assumption and prove it is at least as hard as the traditional DDH assumption. The security of the Sender is reduced to this assumption.

8.3.1.1 *DDH assumption*

Traditionally the DDH assumption states that the tuples (g^x, g^y, g^z) and $(g^x, g^y, g^{x \cdot y})$ are hard to distinguish (where x, y and z are random samples), the variant states that (h, g^r, h^r) and $(h, g^r, h^r \cdot g)$ are hard to distinguish (where $h \in G$, and g is the generator of G). We formalise this variant of the assumption and prove it is as hard as the original DDH assumption in Lemma 39. The variant is formalised using the following two games.

$$\begin{aligned}
&\textit{DDH-}0(\mathcal{A}) = \textit{do } \{\\
&\quad s \leftarrow \textit{samp-uniform}(|G|);\\
&\quad r \leftarrow \textit{samp-uniform}(|G|);\\
&\quad \textit{let } h = g^s;\\
&\quad \mathcal{A}(h, g^r, h^r)\}
\end{aligned} \tag{8.21}$$

$$\begin{aligned}
&\textit{DDH-}1(\mathcal{A}) = \textit{do } \{\\
&\quad s \leftarrow \textit{uniform}(|G|);\\
&\quad r \leftarrow \textit{uniform}(|G|);\\
&\quad \textit{let } h = g^s;\\
&\quad \mathcal{A}(h, g^r, h^r \otimes g)\}
\end{aligned} \tag{8.22}$$

The advantage an adversary has of beating this variant of the DDH assumption is defined as the probability it has of distinguishing the games.

Definition 34 (variant of DDH advantage)**.**

$$\textit{DDH-adv}_{\textit{var}}(\mathcal{A}) = |\mathcal{P}[\textit{DDH-}0(\mathcal{A}) = \textit{True}] - \textit{DDH-}1(\mathcal{A}) = \textit{True}]|$$

We show the variant is at least as hard as the original DDH assumption by bounding the advantage $\textit{DDH-adv}_{\textit{var}}$ from above by the original advantage in the following Lemma.

Lemma 39. shows $\textit{DDH-adv}_{\textit{var}}(\mathcal{A}) \leq \textit{DDH-adv}(\mathcal{A}) + \textit{DDH-adv}(\mathcal{A}'(\mathcal{A}))$

Where *DDH-adv* is the original DDH advantage (formalised in [53], we outline this formalisation in Appendix 7.3.2.1, $\textit{DDH-adv}_{\textit{var}}$ is the definition we make of the advantage of the variant on the DDH assumption and $\mathcal{A}'(D, a, b, c) = D(a, b, c \otimes g)$ is an adversary we construct to interact with the DDH assumption. Lemma 39 shows the variant is at least as hard as the original assumption.

We now show security of each party in turn.

PARTY 1 The real view for party 1 is constructed as follows.

```
$R_1((m_0, m_1), \sigma, z(\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)) = do\ \{$
  $r \leftarrow samp\text{-}uniform(|G|);$
  $\alpha_0 \leftarrow samp\text{-}uniform(|G|);$
  $\alpha_1 \leftarrow samp\text{-}uniform(|G|);$
  $let\ h_0 = g^{\alpha_0};$
  $let\ h_1 = g^{\alpha_1};$
  $let\ a = g^r;$
  $let\ b_0 = h_0^r \otimes g^{(if\ \sigma\ then\ 1\ else\ 0)};$
  $let\ b_1 = h_1^r \otimes g^{(if\ \sigma\ then\ 1\ else\ 0)};$                    (8.23)
  $((in_1, in_2, in_3), s) \leftarrow \mathcal{A}((m_0, m_1), h_0, h_1, a, b_0, b_1, z);$
  $let\ (h, a, b) = (h_0 \otimes inv(h_1), a, b_0 \otimes inv(b_1));$
  $(b, \_) \leftarrow F_{ZK}^{DH}((in_1, in_2, in_3), ((h, a, b), r));$
  $\_ \leftarrow assert(b);$
  $(((w_0, z_0), (w_1, z_1)), s') \leftarrow \mathcal{A}_2(h_0, h_1, a, b_0, b_1, (m_0, m_1), s);$
  $out_1 \leftarrow \mathcal{A}_3(s');$
  $return(out_1, if\ \sigma\ then\ (z_1 \otimes inv(w_1^{\alpha_1}))\ else\ (z_0 \otimes inv(w_0^{\alpha_0})))\}$
```

To show security for party 1 we make a case split on σ. We construct $S_1 = (S_{1,P_1}, S_{2,P_1})$ as given in [44] and construct an adversary for each case ($DDH\text{-}\mathcal{A}_{\sigma=1}$, $DDH\text{-}\mathcal{A}_{\sigma=0}$) that *breaks* the variant of the DDH assumption and prove the reduction in Lemma 40. The components of the simulator are given below.

```
$S_{1,P_1}((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), (m_0, m_1), z) = do\ \{$
  $r \leftarrow samp\text{-}uniform(|G|);$
  $\alpha_0 \leftarrow samp\text{-}uniform(|G|);$
  $\alpha_1 \leftarrow samp\text{-}uniform(|G|);$
  $let\ h_0 = g^{\alpha_0};$
  $let\ h_1 = g^{\alpha_1};$
  $let\ a = g^r;$
  $let\ b_0 = h_0^r$
  $let\ b_1 = h_1^r \otimes g$                    (8.24)
  $((in_1, in_2, in_3), s) \leftarrow \mathcal{A}((m_0, m_1), h_0, h_1, a, b_0, b_1, z);$
  $let\ (h, a, b) = (h_0 \otimes inv(h_1), a, b_0 \otimes inv(b_1));$
  $\_ \leftarrow assert((in_1, in_2, in_3), = (h, a, b));$
  $(((w_0, z_0), (w_1, z_1)), s') \leftarrow \mathcal{A}_2(h_0, h_1, a, b_0, b_1, (m_0, m_1), s);$
  $let\ x_0 = z_0 \otimes (inv(w_0^{\alpha_0}));$
  $let\ x_1 = z_1 \otimes (inv(w_1^{\alpha_1}));$
  $return((x_0, x_1), s')\}$
```

$$S_{2,P_1}((\mathcal{A}_1,\mathcal{A}_2,\mathcal{A}_3),(m_0,m_1),z,out_1) = \mathcal{A}_3(s') \tag{8.25}$$

Lemma 40. *(in mal-ot)*
assumes $\sigma = False$
shows $adv\text{-}P_1((m_0,m_1),\sigma,z,S_1,\mathcal{A},D)) = DDH\text{-}adv_{var}(DDH\text{-}\mathcal{A}_{\sigma=0}((m_0,m_1),z,\mathcal{A},D))$

Proof. We construct the adversary that plays against the variant of the DDH assumption as follows.

```
$DDH\text{-}\mathcal{A}_{\sigma=0}((m_0,m_1),z,(\mathcal{A}_1,\mathcal{A}_2,\mathcal{A}_3),D,h,a,t)$
    $\alpha_0 \leftarrow samp\text{-}uniform(|G|);$
    $let\ h_0 = g^{\alpha_0};$
    $let\ h_1 = h;$
    $let\ b_0 = a^{\alpha_0};$
    $let\ b_1 = t;$
    $((in_1,in_2,in_3),s) \leftarrow \mathcal{A}_1((m_0,m_1),h_0,h_1,a,b_0,b_1,z);$
    $\_ \leftarrow assert(in_1 = h_0 \otimes inv(h_1) \wedge in_2 = a \wedge in_3 = b_0 \otimes inv(b_1));$
    $(((w_0,z_0),(w_1,z_1)),s') \leftarrow \mathcal{A}_2(h_0,h_1,a,b_0,b_1,(m_0,m_1),s);$
    $let\ x_0 = z_0 \otimes (inv(w_0)^{\alpha_0}));$
    $adv\text{-}out \leftarrow \mathcal{A}_3(s');$
    $D(adv\text{-}out,x_0)\}$
```

Here the (h,a,t) represent the tuple it takes as input. The idea behind the proof is to show in the case that (h,a,t) is not DDH tuple, then the output of the adversary is identical to the ideal model and if (h,a,t) is a DDH tuple then the output of the adversary is equal to the real view. Proving this is sufficient to prove the statement after considering the definitions of the two advantages.

The argument of proof runs as follows: if (h,a,t) is a DDH tuple then the adversary generates (h_0,h_1,a,b_0,b_1) in the same way as an honest R would when $\sigma = False$, if (h,a,t) is not a DDH tuple the adversary generates (h_0,h_1,a,b_0,b_1) in the same way as the simulator. Using (h_0,h_1,a,b_0,b_1) the adversary continues the simulation. Finally the output of the adversary is also dependent on whether the input was a DDH tuple or not: if the input is a DDH tuple then output is equal to the output of the real view, and if the input is not a DDH tuple then the output is equal to the simulated view. □

Lemma 41. *(in mal-ot)*
assumes $\sigma = True$
shows $adv\text{-}P_1((m_0,m_1),\sigma,z,\mathcal{A},D)) = DDH\text{-}adv_{var}(DDH\text{-}\mathcal{A}_{\sigma=1}(\mathcal{A},D))$

Proof. The proof is analogous to the proof of Lemma 40. We do not repeat it here, only show the adversary we construct.

```
$DDH\text{-}\mathcal{A}_{\sigma=1}((m_0, m_1), z, (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), D, h, a, t)$
  $\alpha_1 \leftarrow samp\text{-}uniform(|G|);$
  $let\ h_1 = g^{\alpha_1};$
  $let\ h_0 = h;$
  $let\ b_1 = a^{\alpha_1} \otimes g;$
  $let\ b_0 = t;$
  $((in_1, in_2, in_3), s) \leftarrow \mathcal{A}_1((m_0, m_1), h_0, h_1, a, b_0, b_1, z);$
  $\_ \leftarrow assert(in_1 = h_0 \otimes inv(h_1) \wedge in_2 = a \wedge in_3 = b_0 \otimes inv(b_1));$
  $(((w_0, z_0), (w_1, z_1)), s') \leftarrow \mathcal{A}_2(h_0, h_1, a, b_0, b_1, (m_0, m_1), s);$
  $let\ x_1 = z_1 \otimes (inv(w_1)^{\alpha_1}));$
  $adv\text{-}out \leftarrow \mathcal{A}_3(s');$
  $D(adv\text{-}out, x_1)\}$
```

Comparing this to the adversary used in Lemma 40 we notice the only real difference is the ordering on h_0, h_1 and b_0, b_1. □

Using Lemma 39 we bound the malicious advantages by the traditional DDH assumption advantage.

Theorem 34. *(in mal-ot)*
assumes $\sigma = False$
shows $adv\text{-}P_1((m_0, m_1), \sigma, z, \mathcal{A}, D)) \leq$
$DDH\text{-}adv(DDH\text{-}\mathcal{A}_{\sigma=0}(\mathcal{A}, D)) +$
$DDH\text{-}adv(\mathcal{A}'(DDH\text{-}\mathcal{A}_{\sigma=0}(\mathcal{A}, D)))$

Theorem 35. *(in mal-ot)*
assumes $\sigma = True$
shows $adv\text{-}P_1((m_0, m_1), \sigma, z, \mathcal{A}, D)) \leq$
$DDH\text{-}adv(DDH\text{-}\mathcal{A}_{\sigma=1}(\mathcal{A}, D)) +$
$DDH\text{-}adv(\mathcal{A}'(DDH\text{-}\mathcal{A}_{\sigma=1}(\mathcal{A}, D)))$

Together Theorems 34 and 35 show security in the case the Sender is corrupted in Protocol 5. We have shown that in both cases ($\sigma = True$ and $\sigma = False$) the advantage is less than the sum of two DDH advantages, which we assume to be small — and thus their sum is also small.

PARTY 2 The real view for party 2 is constructed as follows.

$$
\begin{aligned}
&R_1((m_0,m_1),\sigma,z(\mathcal{A}_1,\mathcal{A}_2,\mathcal{A}_3)) = \mathit{do}\ \{\\
&\quad ((h_0,h_1,a,b_0,b_1),s) \leftarrow \mathcal{A}_1(\sigma,z);\\
&\quad \_ \leftarrow \mathit{assert}(h_0 \in \mathit{carrier}(G) \wedge h_1 \in \mathit{carrier}(G) \wedge a \in \mathit{carrier}(G) \wedge b_0 \in \mathit{carrier}(G) \wedge b_1);\\
&\quad (((\mathit{in}_1,\mathit{in}_2,\mathit{in}_3),r),s') \leftarrow \mathcal{A}_2((h_0,h_1,a,b_0,b_1),s);\\
&\quad \mathit{let}\ (h,a,b) = (h_0 \otimes \mathit{inv}(h_1),a,b_0 \otimes \mathit{inv}(b_1));\\
&\quad (b,\_) \leftarrow F_{ZK}^{DH}((h,a,b),((\mathit{in}_1,\mathit{in}_2,\mathit{in}_3),r));\\
&\quad \_ \leftarrow \mathit{assert}(b);\\
&\quad u_0 \leftarrow \mathit{samp\text{-}uniform}(|G|);\\
&\quad u_1 \leftarrow \mathit{samp\text{-}uniform}(|G|);\\
&\quad v_0 \leftarrow \mathit{samp\text{-}uniform}(|G|);\\
&\quad v_1 \leftarrow \mathit{samp\text{-}uniform}(|G|);\\
&\quad \mathit{let}\ z_0 = b_0^{u_0} \otimes h_0^{v_0} \otimes m_0;\\
&\quad \mathit{let}\ w_0 = a^{u_0} \otimes g^{v_0};\\
&\quad \mathit{let}\ e_0 = (w_0,z_0);\\
&\quad \mathit{let}\ z_1 = (b_1 \otimes \mathit{inv}(g))^{u_1} \otimes h_1^{v_1} \otimes m_1;\\
&\quad \mathit{let}\ w_1 = a^{u_1} \otimes g^{v_1};\\
&\quad \mathit{let}\ e_1 = (w_1,z_1);\\
&\quad \mathit{out}_2 \leftarrow \mathcal{A}_3(e_0,e_1,s');\\
&\quad \mathit{return}((),\mathit{out}_2)\}
\end{aligned}
\tag{8.26}
$$

To show security for party 2 we construct two simulators, S_{1,P_2}, S_{2,P_2} that interact with the ideal model such that the output distributions of the real and ideal model are equal — that is we show perfect security. The components of the simulator below.

$$
\begin{aligned}
&S_{1,P_2}((\mathcal{A}_1,\mathcal{A}_2,\mathcal{A}_3),\sigma,z) = \mathit{do}\ \{\\
&\quad (h_0,h_1,a,b_0,b_1) \leftarrow \mathcal{A}_1(\sigma);\\
&\quad \_ \leftarrow \mathit{assert}(h_0,h_1,a,b_0,b_1 \in G);\\
&\quad ((\mathit{in}_1,\mathit{in}_2,\mathit{in}_3),r) \leftarrow \mathcal{A}_2(h_0,h_1,a,b_0,b_1);\\
&\quad \mathit{let}\ (h,a,b) = (\tfrac{h_0}{h_1},a,\tfrac{b_0}{b_1});\\
&\quad b,\_) \leftarrow F_{ZK}^{DH}((h,a,b),((\mathit{in}_1,\mathit{in}_2,\mathit{in}_3),r));\\
&\quad \_ \leftarrow \mathit{assert}(b);\\
&\quad \mathit{let}\ l = \tfrac{b_0}{h_0^r};\\
&\quad \mathit{return}((\mathit{if}\ l = 1\ \mathit{then}\ \mathit{False}\ \mathit{else}\ \mathit{True}),(h_0,h_1,a,b_0,b_1))\}
\end{aligned}
\tag{8.27}
$$

$$\begin{aligned}
&S_{2,P_2}((\mathcal{A}_1,\mathcal{A}_2,\mathcal{A}_3),\sigma,z,m_\sigma,((h_0,h_1,a,b_0,b_1),s)) = do\ \{\\
&\quad u_0,v_0,u_1,v_1 \leftarrow uniform(|G|);\\
&\quad v_0 \leftarrow samp\text{-}uniform(|G|);\\
&\quad u_1 \leftarrow samp\text{-}uniform(|G|);\\
&\quad v_1 \leftarrow samp\text{-}uniform(|G|);\\
&\quad ((in_1,in_2,in_3),r) \leftarrow \mathcal{A}_2(h_0,h_1,a,b_0,b_1);\\
&\quad let\ w_0 = a^{u_0} \otimes g^{v_0};\\
&\quad let\ w_1 = a^{u_1} \otimes g^{v_1};\\
&\quad let\ z_0 = b_0^{u_0} \otimes h_0^{v_0} \otimes if\ \sigma\ then\ 1\ else\ m_\sigma);\\
&\quad let\ z_1 = (\tfrac{b_1}{g})^{u_1} \otimes h_1^{v_1} \otimes if\ \sigma\ then\ m_\sigma\ else\ 1);\\
&\quad \mathcal{A}_3((w_0,z_0),(w_1,z_1))\}
\end{aligned} \tag{8.28}$$

To show equality between the real and ideal views we consider the cases on $l = \frac{b_0}{h_0^r}$ (constructed by S_{1,P_2}): $l = 1$, $l = g$, $l \notin \{1, g\}$. Such a case split is easy to reason about on paper, however due to l being defined inside a probabilistic program we cannot easily access it to perform the case split.

To make the case split easier we introduce a *case separation* technique. This allows us to more easily prove a property on a probabilistic program where the cases on a bound variable need to be considered. We describe our method informally here.

We isolate the case splitting variable by defining a new program that replicates the original program from the point the variable is introduced; the variable is now an input to a probabilistic program. Case splitting on inputs is natural so we prove the required property on the new program noting that some contextual assumptions may be needed in the statement, these are seen in the assumptions to Lemma 42. Using this statement we prove the required property on the original probabilistic program by recombining the newly defined program and the property proven about it and the original program.

More specifically we define $mal_{ideal\text{-}end_2}$ that describes the end of the ideal view's probabilistic program, beginning at a point where l can be taken as an input. We also define the analogous end probabilistic program for the real view ($mal_{real\text{-}end_2}$) and show the two programs are equal by case splitting on l.

Lemma 42. *(in mal-ot)*
assumes $\frac{b_0}{b_1} = (\frac{h_0}{h_1})^r$
and $h_0, h_1, b_0, b_1, m_0, m_1 \in G$
shows $mal_{ideal\text{-}end_2}((m_0,m_1), l, (h_0,h_1,g^r,b_0,b_1), \mathcal{A}_3) =$
$mal_{real\text{-}end_2}((m_0,m_1),(h_0,h_1,g^r,b_0,b_1),\mathcal{A}_3)$

Proof. We can make the case split on $l = \frac{b_0}{h_0^r}$ easily. We consider the cases separately here. The proofs for each case are not interesting from a cryptographic standpoint. Thus we point the reader to [43, p 193-195] and our formalisation for a detailed proof. □

Using Lemma 42 we show security for party 2.

Theorem 36. *(in mal-ot)*
assumes $m_0 \in G$
and $m_1 \in G$
shows *perfect-sec*$_2((m_0, m_1), \sigma, z, \mathcal{A})$

Proof. We show the real and ideal views are equal up to the point of l being introduced (recall the ideal view is constructed using the simulator, where l is constructed). This fact, together with Lemma 42, is used to show equality between the views. Thus we have shown security for Protocol 5. □

We note Theorem 36 could be proven without Lemma 42 by manipulating the probabilistic programs, however this would have required more subtleties and low level reasoning. In fact we proved Lemma 34 without our case separation technique to provide a comparison of the methods. Our technique does not necessarily reduce the size of the proof however we believe it does reduce the technicality of the proof.

This section has shown how we instantiate our framework defining malicious security in the two party setting. Completing proofs of malicious security are more complex than proofs in the semi-honest setting. We feel, however, that this complexity is mainly due to the protocols being longer and more elaborate rather than the formalised definitions being more difficult to deal with.

9 CONCLUSION

This thesis was motivated by the potential "crisis of rigor" in cryptography as remarked by Bellare and Rogaway [11]. We choose to begin our formalisation effort with MPC as this is an area of cryptography that is beginning to be widely implemented. Thus it is important to study it under the lens of formal methods. During our study of MPC however it became apparent that the study of low level primitives is also important. Σ-protocols and commitment schemes are often used as building blocks for larger protocols.

We believe the work in this thesis has contributed in two main aspects:

1. We have provided strong foundations for reasoning about modern cryptography. The frameworks we provide are general and abstract, meaning others can use them for further instantiations of proofs or as building blocks in modular proofs — like we have in our general construction of commitment schemes from Σ-protocols and our proof of the two party GMW protocol.

2. Our work has provided further evidence that CryptHOL is a suitable tool for reasoning about reduction based cryptography. Due to the fact CryptHOL is a relatively young framework our proofs contribute heavily to the body of proof conducted thus far.

In the rest of this concluding chapter we first discuss the work most closely related to that in this thesis and then suggest how we believe our work can be extended. Finally we make an appeal to both communities, the formal methods community and the cryptographic community, and suggest how we think they can work together in this area of research.

9.1 RELATED WORK AND DISCUSSION

In this Section we pick out other formalisations of MPC and Σ-protocols and commitment schemes and discuss how they relate to our work.

9.1.1 *MPC*

Almeida et al. [1] define two party semi-honest security for MPC in EasyCrypt. As we describe in Section 7.2.3 they define a game formalise simulation-based security. Our proof that our definitions are equivalent to theirs adds confidence to both sets of formalised definitions. In [1] the authors prove security of Yao's Secure Function Evaluation (SFE) which requires the formalisation of OT^1_2.

The work of Haagh et al. [41] provides definitions for malicious security and proved secure Mauer's protocol [54] for secure addition and multiplication for n parties. The authors also provide security definitions for the semi-honest setting, however do not reuse the definitions from [1]. This was the first work to consider the malicious setting however the definitions captured in the formalisation were not the traditional literature definitions. Instead the authors formalised their own flavour of *non interference* definitions and a meta (proven on paper) theorem that they implied the traditional paper definitions of malicious security — the definitions we consider in our work. Their work improves on our work as they consider the n party setting whereas we only consider the two party setting. The definitions however and proofs they provide are only for perfect security and do not consider the reduction based security we consider here.

9.1.2 Σ-*protocols and commitment schemes*

Commitment schemes have been studied in EasyCrypt by Metere et al. [56] where the Pedersen commitment scheme was proven secure. We also consider the Pedersen commitment scheme. One noticeable difference between the proof effort required in Isabelle is in the construction of the adversary used to prove computational binding — in particular in outputting the inverse of an element in a field. In EasyCrypt the inverse function is defined with the required property, that is: $x \neq 0 \Rightarrow x \cdot inv(x) = 1$ and consequently division is defined as $y \neq 0 \Rightarrow x = x \cdot inv(y)$. In Isabelle on the other hand we do not axiomatise the property of an inverse, but derive it from the Bezout function. This means our approach could be considered more foundational, and thus warrants the extra proof effort required.

Σ-protocols have been considered in [7] using CertiCrypt. Here Barthe et al. first prove secure a general construction of Σ^{ϕ}-protocols that prove knowledge of a preimage under a group homomorphism ϕ — the Schnorr and Okamoto Σ- protocols that we formalise are examples of this type. Secondly they consider the compound AND and OR statements we also formalise. Their work however only considered the compound statements over bitstrings whereas our formalisation is over an arbitrary boolean algebra of which bitstrings of a given length are one instance. While we do not see this as a major contribution we feel the generalisation is useful in highlighting that only the one time pad property for the xor function is required, and thus a boolean algebra suffices.

Both [56] and [7] formalise some of the protocols we consider however they do so in different frameworks. For the ongoing development of the area we believe that it is important to have up-to-date and usable formalisations in the same framework; therefore we feel our work provides a strong basis for further formalisations in this area. In particular, to the best of our knowledge, our work is the first to link Σ-protocols and commitment schemes.

9.1.3 *Extending this work*

This thesis is split into two parts; the formalisation of MPC and the formalisation of Σ-protocols and commitment schemes. At present the two parts are distinct. One concerns protocols that achieve MPC and the other concerns two low level primitives. In reality, however, these two are not distinct. They are heavily related in that Σ-protocols and commitment schemes are used in the construction of many MPC protocols. This is especially relevant in the malicious security model for MPC. Here we must hold parties to account so they do not cheat in the protocol, or if they do, they get caught. In the malicious MPC protocol we have formalised we relied on a Zero-Knowledge functionality that proved a tuple was a DDH tuple.

The first step to linking the two parts of this thesis would be to extend the formalisation of Σ-protocols to capture Zero-Knowledge proofs. Σ-protocols provide a baseline for Zero-Knowledge, in particular they provide the HVZK property. This will allow more malicious MPC protocols to be considered.

Using the frameworks for the primitives (Σ-protocols, ZK proofs and commitment schemes) we envisage being able to "cut and paste" proofs using the modularity available in Isabelle and CryptHOL. By "cut and paste" we mean prove protocols secure assuming the required underlining primitives and constructions exist. In the first instance this will provide an abstract proof that the protocol is secure. One can then prove different instantiations of the underlying primitives secure. Isabelle's module system will then allow the general proof to be easily instantiated for the chosen instantiation of the underlying primitive(s).

To some extent this thesis achieves this "cut and paste" method. We prove the construction of commitment schemes from Σ-protocols at an abstract level, prove various Σ-protocols secure and then instantiate the general proof using the results from the Σ-protocols we consider. In principle this achieve our desired methodology, albeit in a simple version of it.

9.2 APPEAL TO TWO COMMUNITIES

We envisage this thesis has two sets of readers, cryptographers and formal methods experts. For the cryptographer, we hope you can see that proof is often more involved and technical than at first sight. Often trivial things may be overlooked or not considered. A good illustration of this is in the definitions of Σ-protocols — we discussed the many paper definitions in detail in Section 4.2.1. We therefore hope that you view formalisations of cryptography as worthwhile; it may take time and much effort to formalise well known results but every field of research must start somewhere, and stepping stones along the way, however small, are required. For the formal methods expert we hope you can see how the tools and years of research into foundational areas can have impact when applied to settings such as security and that this can motivate you to continue work that oftentimes goes unrewarded outside the community.

Like most areas that are formed from the intersection of two well established fields real progress happens when both sides "put some skin in the game"[1]. For the cryptographers we feel this should be to allow the formalisations time to catch up. Formal methods moves at its own pace and should be allowed to do so. If the cryptographic community indeed does feel like formal methods are important to its continuing development then the community should be willing to engage more with

[1] Jeremy Avigad appealed to mathematicians to "put some skin in the game" [4] when writing about the "Mechanisation of Mathematics".

it; tell the formal methods community exactly what you want from the work. Whether that be complete proofs of complex protocols, which will take time, or proofs of specific *tricky* parts of constructions, parts where cryptographers know their intuition and paper proofs sometimes fail them. The contribution the formal methods community can make is to complete the feedback loop. Study of cryptographic proof under the lens of formal methods often highlights short comings and informal arguments. This must be communicated so future proofs and subsequent work can be improved.

BIBLIOGRAPHY

[1] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, and Vitor Pereira. "A Fast and Verified Software Stack for Secure Function Evaluation." In: *ACM Conference on Computer and Communications Security*. ACM, 2017, pp. 1989–2006.

[2] David W. Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P. Smart, and Rebecca N. Wright. "From Keys to Databases - Real-World Applications of Secure Multi-Party Computation." In: *Comput. J.* 61.12 (2018), pp. 1749–1771.

[3] "Archive of Formal Proofs (AFP)." In: (). `https://www.isa-afp.org`.

[4] Jeremy Avigad. "Mechanization of Mathematics." In: *Notices of the AMS*. 2018, 65(06):681–690,

[5] G Barthe, B Grégoire, and S Zanella Béguelin. "Formal certification of code-based cryptographic proofs." In: *POPL*. ACM, 2009, pp. 90–101.

[6] G Barthe, B Grégoire, S Heraud, and S Zanella Béguelin. "Computer-Aided Security Proofs for the Working Cryptographer." In: *CRYPTO*. Vol. 6841. Lecture Notes in Computer Science. Springer, 2011, pp. 71–90.

[7] Gilles Barthe, Daniel Hedin, Santiago Zanella Béguelin, Benjamin Grégoire, and Sylvain Heraud. "A Machine-Checked Formalization of Sigma-Protocols." In: *CSF*. IEEE Computer Society, 2010, pp. 246–260.

[8] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. "CryptHOL: Game-based Proofs in Higher-order Logic." In: *IACR Cryptology ePrint Archive* (2017), p. 753.

[9] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization." In: *CRYPTO*. Vol. 576. Lecture Notes in Computer Science. Springer, 1991, pp. 420–432.

[10] Mihir Bellare and Phillip Rogaway. "Optimal Asymmetric Encryption." In: *EUROCRYPT*. Vol. 950. Lecture Notes in Computer Science. Springer, 1994, pp. 92–111.

[11] Mihir Bellare and Phillip Rogaway. "Code-Based Game-Playing Proofs and the Security of Triple Encryption." In: *IACR Cryptology ePrint Archive* 2004 (2004), p. 331.

[12] Bruno Blanchet. "A Computationally Sound Mechanized Prover for Security Protocols." In: *IEEE Trans. Dependable Sec. Comput.* 5.4 (2008), pp. 193–207.

[13] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. "Extending Sledgehammer with SMT Solvers." In: *J. Autom. Reasoning* 51.1 (2013), pp. 109–128.

[14] Manuel Blum. "Coin Flipping by Telephone." In: *CRYPTO*. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04, 1981, pp. 11–15.

[15] Manuel Blum. “How to prove a theorem so no one else can claim it.” In: *International Congress of Mathematicians*. 1986, pp. 1444–1451.

[16] Manuel Blum, Paul Feldman, and Silvio Micali. “Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract).” In: *STOC*. ACM, 1988, pp. 103–112.

[17] Carlo Blundo, Barbara Masucci, Douglas R. Stinson, and Ruizhong Wei. “Constructions and Bounds for Unconditionally Secure Non-Interactive Commitment Schemes.” In: *Des. Codes Cryptogr.* 26.1-3 (2002), pp. 97–110.

[18] Dan Bogdanov, Sven Laur, and Jan Willemson. “Sharemind: A Framework for Fast Privacy-Preserving Computations.” In: *ESORICS*. Vol. 5283. Lecture Notes in Computer Science. Springer, 2008, pp. 192–206.

[19] Peter Bogetoft et al. “Secure Multiparty Computation Goes Live.” In: *Financial Cryptography*. Vol. 5628. Lecture Notes in Computer Science. Springer, 2009, pp. 325–343.

[20] David Butler and David Aspinall. “Multi-Party Computation.” In: *Archive of Formal Proofs* (May 2019). https://www.isa-afp.org/entries/Multi_Party_Computation.html, Formal proof development.

[21] David Butler, David Aspinall, and Adrià Gascón. “How to Simulate It in Isabelle: Towards Formal Proof for Secure Multi-Party Computation.” In: *ITP*. Vol. 10499. Lecture Notes in Computer Science. Springer, 2017, pp. 114–130.

[22] David Butler, David Aspinall, and Adrià Gascón. “On the Formalisation of Σ-Protocols and Commitment Schemes.” In: *POST*. Vol. 11426. Lecture Notes in Computer Science. Springer, 2019, pp. 175–196.

[23] David Butler, David Aspinall, and Adrià Gascón. “Formalising oblivious transfer in the semi-honest and malicious model in CryptHOL.” In: *CPP*. ACM, 2020, pp. 229–243.

[24] David Butler and Andreas Lochbihler. “Sigma Protocols and Commitment Schemes.” In: *Archive of Formal Proofs* (2019). https://www.isa-afp.org/entries/Sigma_Commit_Crypto.html, Formal proof development.

[25] David Butler, Andreas Lochbihler, David Aspinall, and Adrià Gascón. “Formalising Σ-Protocols and Commitment Schemes using CryptHOL.” In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 1185.

[26] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols.” In: *FOCS*. IEEE Computer Society, 2001, pp. 136–145.

[27] David Chaum and Torben P. Pedersen. “Wallet Databases with Observers.” In: *CRYPTO*. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 89–105.

[28] Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. “Improved OR-Composition of Sigma-Protocols.” In: *Theory of Cryptography*. Ed. by Eyal Kushilevitz and Tal Malkin. Springer, 2016, pp. 112–141.

[29] R. Cramer. “Modular Design of Secure, yet Practical Cryptographic Protocols.” In: *PhD thesisPhD Thesis, University of Amsterdam* (1996).

[30] Ronald Cramer and Victor Shoup. “A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack.” In: *CRYPTO*. Vol. 1462. Lecture Notes in Computer Science. Springer, 1998, pp. 13–25.

[31] I. Damgard. “On Σ-protocols.” In: *Lecture Notes, University of Aarhus, Department for Computer Science.* (2002).

[32] Ivan Damgård. “On the Existence of Bit Commitment Schemes and Zero-Knowledge Proofs.” In: *CRYPTO*. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 17–27.

[33] Whitfield Diffie and Martin E. Hellman. “New directions in cryptography.” In: *IEEE Trans. Information Theory* 22.6 (1976), pp. 644–654.

[34] Shimon Even. “Protocol for Signing Contracts.” In: *CRYPTO*. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04, 1981, pp. 148–153.

[35] Shimon Even, Oded Goldreich, and Abraham Lempel. “A Randomized Protocol for Signing Contracts.” In: *Commun. ACM* 28.6 (1985), pp. 637–647.

[36] Taher El Gamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.” In: *CRYPTO*. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 10–18.

[37] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[38] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[39] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority.” In: *STOC*. ACM, 1987, pp. 218–229.

[40] Michael J. C. Gordon, Robin Milner, L. Morris, Malcolm C. Newey, and Christopher P. Wadsworth. “A Metalanguage for Interactive Proof in LCF.” In: *POPL*. ACM Press, 1978, pp. 119–130.

[41] Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. “Computer-Aided Proofs for Multiparty Computation with Active Security.” In: *CSF*. IEEE Computer Society, 2018, pp. 119–131.

[42] Shai Halevi. “A plausible approach to computer-aided cryptographic proofs.” In: *IACR Cryptology ePrint Archive* 2005 (2005), p. 181.

[43] Carmit Hazay and Yehuda Lindell. “A Note on the Relation between the Definitions of Security for Semi-Honest and Malicious Adversaries.” In: *IACR Cryptology ePrint Archive* 2010 (2010), p. 551.

[44] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.

[45] “Isabelle Distribution Library.” In: `https://isabelle.in.tum.de/library/`.

[46] Neal Koblitz and Alfred Menezes. “Another Look at "Provable Security".” In: *J. Cryptology* 20.1 (2007), pp. 3–37.

[47] John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. "Application-Scale Secure Multiparty Computation." In: *ESOP*. Vol. 8410. Lecture Notes in Computer Science. Springer, 2014, pp. 8–26.

[48] Yehuda Lindell. "How To Simulate It - A Tutorial on the Simulation Proof Technique." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 46.

[49] Yehuda Lindell. "How to Simulate It - A Tutorial on the Simulation Proof Technique." In: *Tutorials on the Foundations of Cryptography*. Springer International Publishing, 2017, pp. 277–346.

[50] Andreas Lochbihler. "Probabilistic Functions and Cryptographic Oracles in Higher Order Logic." In: *ESOP*. Vol. 9632. Lecture Notes in Computer Science. Springer, 2016, pp. 503–531.

[51] Andreas Lochbihler. "CryptHOL." In: *Archive of Formal Proofs* (2017).

[52] Andreas Lochbihler and S. Reza Sefidgar. "A tutorial introduction to CryptHOL." In: https://eprint.iacr.org/2018/941. 2018.

[53] Andreas Lochbihler, S. Reza Sefidgar, and Bhargav Bhatt. "Game-based cryptography in HOL." In: *Archive of Formal Proofs* 2017 ().

[54] Ueli M. Maurer. "Secure multi-party computation made simple." In: *Discrete Applied Mathematics* 154.2 (2006), pp. 370–381.

[55] Ueli Maurer. "Constructive Cryptography - A New Paradigm for Security Definitions and Proofs." In: *TOSCA*. Vol. 6993. Lecture Notes in Computer Science. Springer, 2011, pp. 33–56.

[56] Roberto Metere and Changyu Dong. "Automated Cryptographic Analysis of the Pedersen Commitment Scheme." In: *MMM-ACNS*. Vol. 10446. Lecture Notes in Computer Science. Springer, 2017, pp. 275–287.

[57] Moni Naor and Benny Pinkas. "Efficient oblivious transfer protocols." In: (2001), pp. 448–457.

[58] Tobias Nipkow and Gerwin Klein. *Concrete Semantics - With Isabelle/HOL*. Springer, 2014.

[59] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Vol. 2283. Lecture Notes in Computer Science. Springer, 2002.

[60] Abderrahmane Nitaj and Tajjeeddine Rachidi. "Factoring RSA Moduli with Weak Prime Factors." In: *C2SI*. Vol. 9084. Lecture Notes in Computer Science. Springer, 2015, pp. 361–374.

[61] Eiji Okamoto. "Key Distribution Systems Based on Identification Information." In: *CRYPTO*. Vol. 293. Lecture Notes in Computer Science. Springer, 1987, pp. 194–202.

[62] Lawrence C. Paulson, Tobias Nipkow, and Makarius Wenzel. "From LCF to Isabelle/HOL." In: *CoRR* abs/1907.02836 (2019).

[63] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing." In: *CRYPTO*. Vol. 576. Lecture Notes in Computer Science. Springer, 1991, pp. 129–140.

[64] A Petcher and G Morrisett. "The Foundational Cryptography Framework." In: *POST*. Vol. 9036. Lecture Notes in Computer Science. Springer, 2015, pp. 53–72.

[65] Michael O. Rabin. "How To Exchange Secrets with Oblivious Transfer." In: *IACR Cryptology ePrint Archive* 2005 (2005), p. 187.

[66] Ronald Rivest. "Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer." In: *Unpublished manuscript* (1999).

[67] Joshua Schneider, Manuel Eberl, and Andreas Lochbihler. "Monad normalisation." In: *Archive of Formal Proofs* 2017 ().

[68] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards." In: *J. Cryptology* 4.3 (1991), pp. 161–174.

[69] Adi Shamir. "How to Share a Secret." In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[70] Victor Shoup. "OAEP Reconsidered." In: *IACR Cryptology ePrint Archive* 2000 (2000), p. 60.

[71] Victor Shoup. "Sequences of games: a tool for taming complexity in security proofs." In: *IACR Cryptology ePrint Archive* 2004 (2004), p. 332.

[72] Nigel P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Available at https://www.cs.umd.edu/~waa/414-F11/IntroToCrypto.pdf. Springer, 2016.

[73] Markus Wenzel. "Isabelle, Isar - a versatile environment for human readable formal proof documents." PhD thesis. Technical University Munich, Germany, 2002.

[74] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)." In: *FOCS*. IEEE Computer Society, 1982, pp. 160–164.

[75] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets (Extended Abstract)." In: *FOCS*. IEEE Computer Society, 1986, pp. 162–167.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

`https://bitbucket.org/amiede/classicthesis/`

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

`http://postcards.miede.de/`

Thank you very much for your feedback and contribution.

Final Version as of June 23, 2020 (classicthesis v4.6).