Formalising oblivious transfer in the semi-honest and malicious model in CryptHOL

Citation for published version:
Butler, D, Aspinall, D & Gascon, A 2020, Formalising oblivious transfer in the semi-honest and malicious model in CryptHOL. in J Blanchette & C Hritcu (eds), *CPP 2020 - Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, co-located with POPL 2020.* CPP 2020 - Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, co-located with POPL 2020, Association for Computing Machinery, Inc, pp. 229-243, 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, co-located with POPL 2020, New Orleans, United States, 20/01/20. https://doi.org/10.1145/3372885.3373815

Digital Object Identifier (DOI):
10.1145/3372885.3373815

Published In:
CPP 2020 - Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, co-located with POPL 2020

Formalising Oblivious Transfer in the Semi-Honest and Malicious Model in CryptHOL

David Butler
The Alan Turing Institute
London, UK
dbutler@turing.ac.uk

David Aspinall
The University of Edinburgh
Edinburgh, UK
david.aspinall@ed.ac.uk

Adrià Gascón
Google
London, UK
adriagascon@gmail.com

Abstract

Multi-Party Computation (MPC) allows multiple parties to compute a function together while keeping their inputs private. Large scale implementations of MPC protocols are becoming practical thus it is important to have strong guarantees for the whole development process, from the underlying cryptography to the implementation. Computer aided proofs are a way to provide such guarantees.

We use CryptHOL [5, 26] to formalise a framework for reasoning about two party protocols using the security definitions for MPC. In particular we consider protocols for 1-out-of-2 Oblivious Transfer (OT_2^1) — a fundamental MPC protocol — in both the semi-honest and malicious models. We then extend our semi-honest formalisation to OT_4^1 which is a building block for our proof of security for the two party GMW protocol — a protocol that can securely compute any Boolean circuit.

The semi-honest OT_2^1 protocol we formalise is constructed from Extended Trapdoor Permutations (ETP), we first prove the general construction secure and then instantiate for the RSA collection of functions — a known ETP. Our general proof assumes only the existence of ETPs, meaning any instantiated results come without needing to prove any security properties, only that the requirements of an ETP are met.

Keywords Multi-Party Computation, Oblivious Transfer, Formal Verification, Isabelle/HOL, Malicious Security

1 Introduction

The area of provable security provides a firm mathematical foundation for reasoning about cryptography resulting in proofs of security being at the heart of modern cryptography. However even with the increased rigour and detail this affords cryptographic proofs often still present informal or *intuitive* reasoning.

To overcome the *crisis of rigor* [6] formal tools have been developed to allow for the computer-aided checking of proofs. CryptoVerif [7], CertiCrypt [3], EasyCrypt [4], FCF [32] and CryptHOL [5, 26] have all been used to formalise cryptography. Proofs of security can broadly be partitioned into one of two types: game-based [6, 34] and simulation-based proof [15, 19]. All the tools in the preceding list were initially designed for game-based proofs however some have been used for simulation-based proofs too; in [11] and [21] standalone MPC protocols were considered, whereas more recent work [28] and [16] considers composibility in the form of Constructive Cryptography and Universal Composibility respectively.

In this work we consider the simulation-based paradigm in the context of MPC. MPC aims to provide protocols for parties who wish to jointly compute functions over their inputs while keeping their inputs private. Work on MPC can be traced to Yao [35] 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 [9], secure email filtering and teleconferencing [23] and private statistical computation, e.g., using Sharemind [8]. It is this potential large scale implementation of MPC that heightens the need to examine it under the lens of formal verification.

In this work, we first consider Oblivious Transfer (OT), a two party protocol, which is at the heart of many MPC protocols. Its most simple form is *1-out-of-2 OT* (OT_2^1) where the Receiver chooses to learn one of two pieces of information held by the Sender, and learns nothing of the other piece, moreover the Sender does not learn the Receivers choice. OT_2^1 is described by the *functionality* $f_{OT_2^1}$,

$$f_{OT_2^1}((m_0, m_1), \sigma) = (\_, m_\sigma). \tag{1}$$

Here party 1, the Sender, holds two messages (m_0, m_1) and party 2, the Receiver, holds its choice bit σ. At the end of the protocol the Sender receives nothing (denoted by '_') and the Receiver obtains its chosen message, m_σ.

Our study of OT here is motivated by its almost universal use across MPC and its central role to Garbled Circuits [36] and GMW [20]. The GMW protocol allows for the secure computation of any function that can be represented as a Boolean circuit. To achieve this OT_4^1 is required and thus motivates our formalisation of OT_4^1 as a stepping stone in our formalisation of the two party GMW protocol.

There are two adversary models of differing strength in the simulation-based paradigm. First, the semi-honest model assumes the parties do not deviate from the protocol description. This may appear to be a weak definition, but it ensures

there is no inadvertent data leakage from the protocol and acts as an important baseline of security. For example, if the server of one of the parties was compromised and an adversary accessed the trace of the protocol execution the adversary can learn nothing of the party's inputs beyond what can be learnt from the output. We follow the exposition of Lindell [24] in the semi-honest setting. The second, stronger, model is the malicious model where we allow the adversary to fully control (corrupt) one of the parties. Here we formalise the definitions of malicious security from Goldreich [19] and Lindell and Hazay [22].

We formalise our proofs in the theorem prover Isabelle/HOL using CryptHOL [5, 26] which provides a shallow embedding of a probabilistic programming language. We follow the literature definitions and define security using probabilistic programs that describe the *real view* a party sees in a protocol execution and an ideal world where security is guaranteed by construction. An increasing number of proofs have been completed using CryptHOL [2, 13, 27, 29].

1.1 Contributions

By using the modularity and expressiveness of CryptHOL and Isabelle we construct a framework for reasoning formally about the security of Multi-Party Computation protocols in the two party setting. In this work we instantiate our definitional framework to reason about Oblivious Transfer protocols and the two party GMW protocol.

We believe our work advances the state of the art in two separate directions:

(i) We extend the work of Butler et al. [11] by giving more modular proofs of OT protocols in the semi-honest setting, building OT_4^1 from OT_2^1, and proving security of the two party GMW protocol.

(ii) Moreover we investigate the malicious security model for MPC by proving the OT_2^1 of [22]. This complements the work of Haagh et al. [21] in that the formalisation considers the same security definitions. However our formal proof is for the full simulation-based definition, rather than an intermediate non-interference based definition that was formalised in EasyCrypt and thus we lack the need for any paper proofs.

The motivation for studying the two party GMW protocol is clear, it is a powerful result that allows for the secure computation of any boolean circuit. To study this we require both OT_2^1 and OT_4^1. Our motivation for studying the particular OT_2^1 protocols we choose is twofold: first both OT_2^1 protocols (one in the semi-honest model and one in the malicious model) have proofs presented by Lindell (the latter paper proof is from the book by Lindell and Hazay) [22, 24]. Out of the many proofs written of cryptographic protocols we find his among the clearest and most detailed, and thus this seems like a good starting point for formalisation. Second, the OT_2^1 protocol constructed from an ETP allows us to show how the general proof can be instantiated for the RSA function — we believe this shows the benefits of formalisation as our result allows more instantiations to be proved secure without the need to consider the security proofs in anyway.

Extending our work to maliciously secure OT_4^1 and GMW is left as future work. From our experience of formalising proofs in both semi-honest and malicious models we believe a full formalisation of malicious GMW would require another major proof effort. In particular, one would need to expand the formalisation of Σ-protocols from [12, 14] to Zero Knowledge protocols and work in the n party setting which is thus far not considered in CryptHOL.

All definitions and theorems presented in this paper have been checked by the Isabelle/HOL proof assistant. In addition all statements made in this paper are only slight adaptations from the Isabelle statements, we only slightly modify their syntax for ease of reading. We believe that if the reader can parse the statements presented in the paper then they would be able to parse the formal statements in our theory.

1.2 Related work

Semi-honest security has been considered in EasyCrypt [1] where the security of Garbled circuits is considered. The authors give a formal definition of simulation-based security using a game defined as a probabilistic program. As described in Section 3.1 we prove their definitions are equivalent to ours. The challenge faced in formal verification is to provide definitions that are equivalent to the paper definitions. By showing our definitions are equivalent to the ones provided in [1] we add confidence to this equivalence.

The malicious model we consider has been formally studied in [21] where the authors prove security of Maurer's multiplication protocol [30]. This was the first work to consider malicious security however their formalised definitions were not directly from the literature. The authors proved a meta-theorem (proven on paper) which showed their formalised definitions implied the traditional definitions of malicious security. This is not necessarily a weakness of the work because by proving the meta theorem a new approach is proposed; nonetheless one would prefer a completely formalised approach.

Originally CryptHOL was used for game-based proofs [29] and has recently been used for constructive cryptography [27, 28] and commitment schemes and Σ-protocols [12–14]. In [10, 11] the authors used CryptHOL for MPC protocols in the semi-honest model, this work builds on their definitions to make them more abstract and reusable.

To the best of our knowledge none of the protocols considered in this paper have been formalised in any theorem prover.

1.3 Outline of Paper and Formalisation

An outline of the paper can be seen in Fig. 1. Here dashed boxes represent abstract definitional theories — we provide some proof at this abstract level, for example our proof of equivalence with the definitions of security from [1] in the semi honest setting. Solid boxes represent proofs of security and arrows represent imported theories. We provide formalisation for the whole of Fig. 1.

We outline the work in this paper based on which adversary model (semi-honest or malicious) it corresponds to.

Semi-honest model

- We first formalise a framework for reasoning about the simulation-based definitions for the security of two party MPC protocols (Section 3.1).
- We instantiate our framework to consider a OT_2^1 protocol [17] constructed from a general Extended Trapdoor Permutation (ETP). We prove the construction secure at an abstract level, assuming only the existence of an ETP, meaning the results for known ETPs come for free once we have proved they are in fact ETPs. As a case study we show that RSA forms an ETP and thus prove the corresponding security results (Section 4).
- We prove how OT_4^1 is constructed from OT_2^1. Our proofs here, again, are modular in the sense we assume an OT_2^1 protocol exists to construct OT_4^1. This allows for different OT_2^1 protocols to be *plugged in* at a later time (Appendix C).
- Finally we prove security of AND and XOR gates in the two party GMW protocol. Again we consider this in a modular way by assuming that a secure OT_4^1 protocol exists (Section 5).

Malicious model

- In an analogous way to the semi-honest model we formalise the security definitions for two party MPC protocols. (Section 6)
- We demonstrate how our framework can be instantiated by proving security of a OT_2^1 protocol from [22]. (Section 6)

In our formalisation we first consider security in the concrete setting. Here we assume a constant security parameter is implicit in all algorithms that parametrise the framework. We prove all security notions in this setting first, by showing a reduction for example, before utilising Isabelle's module system to prove security in the asymptotic setting — here we reason about negligible functions in the security parameter. More details about this part of our formalisation are given in Section 4.5.

2 CryptHOL and Isabelle Background

In this section we follow [12] and briefly introduce Isabelle and the notions we use throughout and then highlight and discuss some important aspects of CryptHOL. For more detail on CryptHOL see [5, 26]. The full CryptHOL formalisation is available at [25].

2.1 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 λ-calculus. To indicate that term t has type τ we write $t\ ::\ \tau$. Isabelle uses the symbol $\Rightarrow$ for the function type, so $\alpha \Rightarrow \beta$ is the type of functions that takes an input of type a and has output of type β. The type $'a$ denotes an abstract type. The implication arrow $\longrightarrow$ is used to separate assumptions from conclusions inside a closed HOL statement. Sets, of type $\alpha\ set$ are isomorphic to predicates, of type $\alpha \Rightarrow bool$ via the membership map $\in$. We write $\otimes$ to represent multiplication in a group and $\cdot$ for multiplication of natural numbers.

2.2 CryptHOL

CryptHOL [5, 26] is a framework for reasoning about *reduction-based* security arguments that is embedded inside the Isabelle/HOL theorem prover. At a high level it allows the prover to reason about security by writing probabilistic programs and determine the relationships between them.

CryptHOL, like much of modern cryptography, is based on probability theory. Probabilistic programs in CryptHOL are shallowly embedded as sub-probability 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 for the modelling of 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 modelled using monads. A monad consists of a (polymorphic) type constructor, in this case spmf and two (polymorphic) operations:

- $return\ ::\ \alpha \Rightarrow \alpha\ spmf$
- $bind\ ::\ \alpha\ spmf \Rightarrow\ (\alpha \Rightarrow \beta\ spmf) \Rightarrow \beta\ spmf.$

We now introduce parts of CryptHOL that are particularly relevant to our work.

2.2.1 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, Haskell-style do notation to write probabilistic programs where $do\{x \leftarrow p;\ f(x)\}$ is the probabilistic program that samples from the distribution p and returns the *spmf* produced by f. The do notation desugars to $p \triangleright (\lambda x.\ f(x))$, where $\triangleright$ is the binding operator. The desugaring can be read as; the output of f on input x where x has been sampled from p. We can also return an *spmf* using the monad operation *return*. For example to define the

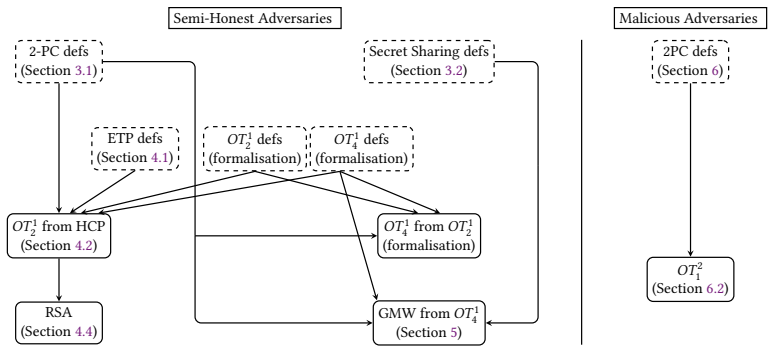

Figure 1. Outline of the formalisation for the paper.

security property of a Hard Core Predicate (HCP) associated with an ETP we define the probabilistic program HCP_{game}.

$$\begin{aligned} &HCP_{game}(\mathcal{A}, \sigma, b_\sigma, D) \equiv do\ \{ \\ &\quad (\alpha, \tau) \leftarrow I; \\ &\quad x \leftarrow S(\alpha); \\ &\quad let\ b = B(\alpha, F^{-1}(\alpha, \tau, x)); \\ &\quad b' \leftarrow \mathcal{A}(\alpha, \sigma, b_\sigma, x, D); \\ &\quad return(b = b')\} \end{aligned} \tag{2}$$

is tasked with guessing b by outputting b', if $b = b'$ we say the adversary wins the game. We introduce this game along with the notion of ETPs in more detail in Section 4.1.

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.

2.2.2 Sampling

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

2.2.3 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. For example, to define the security property of an HCP we consider the probability (advantage) that an adversary wins the HCP game.

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

We provide this definition in full in Definition 5. 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 properties are defined.

2.2.4 Module System

CryptHOL extensively uses the module system available in Isabelle — called locales. 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. In our case locales allow us to define properties of security relative to fixed parameters of abstract type and then instantiate these definitions for explicit protocols and prove the security properties as theorems.

2.2.5 Concrete vs Asymptotic security

In our formalisation, we first prove concrete security bounds using reduction-style proofs. That is, we give a bound on the adversary's advantage (with respect to a security property) as a function of advantages of different adversaries of the primitives used in the construction.

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 Section 4.5 how we achieve this with little effort using Isabelle's locale system.

2.2.6 Negligible functions

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

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

where o is the little o notation. We discuss the use of such functions in our proofs in Section 4.5.

3 Semi-honest security for MPC

In this section we show how we formalise the definitions of security in the semi-honest model and how we define

secret sharing schemes and their correctness. As we are interested in OT we consider only the two party setting. Before detailing our formalisation we first introduce the definitions of security from [24] that we follow.

3.1 Two party protocol security

Definitions from [24] A functionality is a function that maps inputs to desired outputs for a defined protocol problem, for example the functionality describing OT_2^1 is given in Equation 1. Functionalities can be deterministic or non-deterministic[1]. In this section we show our formalisation for the case where the functionality is deterministic (OT is deterministic); for the case where the functionality is non-deterministic we must extend the views in the real and ideal world to also include the output of the protocol — we provide these extended definitions in our formalisation.

For the deterministic case we first require correctness, that is the output of the protocol must equal the output of the functionality.

Intuitively we say a protocol is secure if whatever can be computed by a party can also be *simulated* from only the input and output of the party — in particular simulated not using the input from the other party. We require that the output of the real view and the simulator are indistinguishable. This simulation of the real execution of the protocol means no information is leaked during its execution.

To define security we consider the real/ideal world paradigm. Let π be a two party protocol with inputs (x, y). The real view of the ith party (here $i \in \{1, 2\}$) is denoted by

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

where $w \in \{x, y\}$ and is dependent on which view we are considering, 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. In short the real view is the trace of the execution of the protocol that can be seen by the party.

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,

$$\{S_1(x, f_1(x, y))\} \overset{c}{\equiv} \{view_1^{\pi}(x, y)\} \tag{3}$$

$$\{S_2(y, f_2(x, y))\} \overset{c}{\equiv} \{view_2^{\pi}(x, y)\} \tag{4}$$

where $\overset{c}{\equiv}$ denotes computational indistinguishability. Unfortunately, CryptHOL cannot reason about computational aspects, due to the shallow embedding. We therefore cannot formalise the notion of computational indistinguishability. Instead, we capture the underlying reduction argument in a reduction-based security theorem. For example, in Theorem 3 we bound the advantage a distinguisher has of distinguishing the real and simulated views for party two of the ETP OT_2^1 construction by the advantage an adversary we construct ($\mathcal{A}_{HCP}$) has against the hard core predicate game. The HCP game is considered hard, therefore the reduction implies security.

Reduction-based statements like this capture the key aspects of the security proof and are more generic in the sense that we need not commit to a particular computational model or complexity class such as polynomial time however the reader must manually check that the reduction lies in the desired complexity class.

Formalisation To formalise the semi-honest security definitions we begin by fixing the required parameters to make our definitions in a locale.

locale *semi-honest-det* =
  fixes $funct :: `msg_1 \Rightarrow `msg_2 \Rightarrow (`out_1 \times `out_2)\ spmf$
  and $protocol :: `msg_1 \Rightarrow `msg_2 \Rightarrow (`out_1 \times `out_2)\ spmf$
  and $R_1 :: `msg_1 \Rightarrow `msg_2 \Rightarrow `view_1\ spmf$
  and $S_1 :: `msg_1 \Rightarrow `out_1 \Rightarrow `view_1\ spmf$
  and $R_2 :: `msg_1 \Rightarrow `msg_2 \Rightarrow `view_2\ spmf$
  and $S_2 :: `msg_2 \Rightarrow `out_2 \Rightarrow `view_2\ spmf$

We fix *funct* and *protocol* to represent the probabilistic programs defining the output of the required functionality and the protocol respectively and R_1, S_1, R_2, S_2 to represent the real and simulated views of the parties. In any instantiation it is the provers job to correctly translate the protocol onto the definitions of these parameters.

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

Definition 1 (Correctness)**.**

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

Here m_1 and m_2 are the inputs to the protocol for Party 1 and 2.

In the case of OT_2^1 the functionality given in Equation 1 is encoded in Isabelle as.

$$funct_{OT_2^1}((m_0, m_1), \sigma) = return(\_, if\ \sigma\ then\ m_1\ else\ m_0) \tag{5}$$

Later, in Protocol 1, we will see a protocol that realises this functionality.

Recall that the simulator for party $i \in \{1, 2\}$ takes as input the input for party i and the output from the functionality of party i. We call the output of the simulator on these inputs the ideal view. To construct the ideal view we sample from the functionality and use the binding operator ($\triangleright$) to hand the appropriate output to the simulator. For party 1 the ideal view is as follows:

$$ideal_1(m_1, m_2) = funct(m_1, m_2) \triangleright (\lambda(o_1, o_2).\ S_1(m_1, o_1)) \tag{6}$$

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 (o_1) that has been sampled from the

[1]Here a deterministic functionality will always produce the same output from the same inputs.

functionality. More explicitly, using the monadic do notation this reads:

$$do\ \{(o_1, o_2) \leftarrow funct(m_1, m_2); S_1(m_1, o_1)\}.$$

For perfect security we require the real and simulated views to be equal. We define this for Party 1 below:

Definition 2 (Perfect security, 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.

When perfect security cannot be proven we reason about the advantage a distinguisher has of distinguishing the real and ideal views. We define the advantage of a distinguisher, D, for Party 1 as follows.

Definition 3.

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

The definitions in this section have been extracted from [11] and formalised in a more modular way so they can be instantiated easily for any two party protocol that is considered under this security paradigm.

In Almeida et al. [1] define semi-honest security using a game where a bit is flipped to determine which view the distinguisher is given. As well as the security definitions we provide above, we also define in Isabelle the definitions from [1] and prove the two are equivalent[2].

3.2 Secret sharing schemes

Secret sharing schemes [33] are at the core of MPC protocols. We formalise such schemes by fixing three parameters in a locale; *share*, *reconstruct* and *evaluate*. The first two define the sharing scheme and a third represents the set of functions we wish to realise (in our instantiation of the two party GMW protocol these are AND and XOR). We give their types below.

$$share\ ::\ `a \Rightarrow (`share \times `share)\ spmf \tag{7}$$

$$reconstruct\ ::\ (`share \times `share) \Rightarrow `a\ spmf \tag{8}$$

$$evaluate\ ::\ (`a \Rightarrow `a \Rightarrow `a\ spmf)\ set \tag{9}$$

A sharing scheme is correct if reconstructing a shared input returns the original input.

Definition 4 (Correctness on secret sharing)**.**

$$correct_{share}(input) \equiv (share(input) \rhd (\lambda(s_1, s_2).reconstruct(s_1, s_2)) = return(input))$$

We use the notation given in this Section for the views and advantages throughout the paper however we add subscripts to note which protocol we are considering for clarity.

[2]We do not provide any proof directly from the EasyCrypt definitions, only transcribe their definitions into Isabelle and prove they are equivalent to our definitions.

4 1-out-of-2 OT using Enhanced Trapdoor Permutations

In this section we present our formalisation of the protocol realising OT_2^1 using an Enhanced Trapdoor Permutation (ETP) [17].

4.1 ETPs and HCPs

We recap the paper based definitions of an ETP and refer the reader to [24] (Section 4.3) and [19] (Appendix C.1) for more details.

A collection of trapdoor permutations is a set of permutations (f_α) along with four algorithms I,S,F and F^{-1}, such a collection can be thought of as a collection of one way permutations with a trapdoor such that the inverse can be obtained easily.

- I samples an index α of a permutation, f_α, as well as a corresponding trapdoor τ for the permutation, $(\alpha, \tau) \leftarrow I$.
- $S(\alpha)$ samples a uniform element in the domain of f_α.
- F performs the mapping of f_α, $F(\alpha, x) = f_\alpha(x)$.
- F^{-1} computes the inverse of f_α, $F^{-1}(\alpha, \tau, y) = f_\alpha^{-1}(y)$.

The definition of S provided in [24] and [19] gives 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.

Example 1 (RSA ETP)**.** The RSA function provides an example of and ETP. We informally describe it here and note that we treat it formally in Section B. Here we introduce the ETP as it is done in [18].

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

$$F_{RSA}((N, e), x) = x^e \ mod\ N \tag{10}$$

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

$$F_{RSA}^{-1}((N, e), d, y) = y^d \ mod\ N. \tag{11}$$

Using the trapdoor we correctly compute the inverse, such that the following holds:

$$F_{RSA}^{-1}((N, e), d, F_{RSA}((N, e), x)) = x \tag{12}$$

The range and domain of the RSA ETP are the same, namely $\{0, \ldots N-1\}$ and $S(N, e)$ outputs a uniform sample from this set.

ETPs have the added property that an HCP, B, is associated with it. We assume such a B exists and fix it in the locale. Informally, 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.

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

```
locale etp-base =
  fixes I :: ('index × 'trap) spmf
    and domain :: 'index ⇒ 'domain set
    and range :: 'index ⇒ 'range set
    and F :: 'index ⇒ ('domain ⇒ 'range)
    and F⁻¹ :: 'index ⇒ 'trap ⇒ ('range ⇒ 'domain)
    and B :: 'index ⇒ 'range ⇒ bool
  assumes (α, τ) ∈ set-spmf(I) → domain(α) = range(α)
    and (α, τ) ∈ set-spmf(I) → finite(range(α))
    and (α, τ) ∈ set-spmf(I) → range(α) ≠ {}
    and (α, τ) ∈ set-spmf(I) → bij-betw(F(α), domain(α), range(α))
    and (α, τ) ∈ set-spmf(I) → x ∈ range(α)
            → F⁻¹(α, τ, (F(α, x))) = x
    and lossless-spmf(I)
```

We make six assumptions in the locale on the fixed parameters, the most interesting are the fourth and fifth. The fourth assumption requires that F is a bijection if α is the index outputted by I and the fifth requires that the inverse is correct, namely that applying F^{-1} to F returns the original input. We make this assumption as we cannot explicitly define F^{-1} in general, in particular the inverse function must use the trapdoor τ. We note that F^{-1} must be efficiently computable however as we do not have a notion for runtime in CryptHOL any instantiation of F^{-1} must be manually checked to be computable. In our instantiation for the RSA function the inverse is clearly efficiently computable as it only requires an exponentiation, $F^{-1}_{RSA}((N, e), \tau, y) = y^{\tau} \ mod \ N$.

We define S as uniformly sampling from the range,

$$S(\alpha) = \textit{samp-uniform}(\textit{range}(\alpha)).$$

To formally define the security property of HCPs, we define the HCP advantage adv_{HCP} which captures the probability that $\mathcal{A}$ wins the HCP game. The aim of the adversary $\mathcal{A}$ in the game is to guess the value of B.

Definition 5. *To define adv_{HCP} we use the probabilistic program, HCP_{game}, given in Equation 2 to define the HCP advantage as,*

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

In the HCP game (Equation 2) $\mathcal{A}$ receives α, σ (the Receivers input) and b_σ (the output received by the Receiver) as input. In addition, we must pass x to $\mathcal{A}$. This is because we do not carry around the randomness given to S however we must allow the adversary access to x.

4.2 Realising OT^1_2 using ETPs

We consider the OT^1_2 protocol from [17] which is described in Protocol 1.

Protocol 1. *P_1 has input $(b_0, b_1) \in \{0, 1\}$, P_2 has input $\sigma \in \{0, 1\}$.*

1. *P_1 samples an index and trapdoor, $(\alpha, \tau) \leftarrow I$, 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 of them. 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 have the same distribution.

We formalise the execution of the protocol with the following probabilistic program.

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

Using this definition and the functionality given in Eq. 5 we show correctness of Protocol 1.

Theorem 2.

$$\textit{protocol}_{OT^1_2,ETP}((b_0, b_1), \sigma) = \textit{funct}_{OT^1_2}((b_0, b_1), \sigma)$$

Proofs of correctness are 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. For example we had to prove certain constructed terms are elements of the group when proving correctness of Protocol 4.

4.3 Proving security

To show the protocol is secure in the semi-honest model we consider each party in turn and construct an appropriate simulator. Here we focus on the proof of security for Party 2 as it is more interesting from both a cryptographic and

formal methods point of view. We follow the paper proof from [24] (Section 4.3).

The real view for party 2 is everything the party sees in the execution of the protocol, in this instance that is: the input σ, the index α and the messages from Party 1 $(\beta_\sigma, \beta_{1-\sigma})$. The samples x_σ and $y_{1-\sigma}$ are not included as S is modelled as a probabilistic program. The real view is described by the following probabilistic program.

$$
\begin{aligned}
&R_{2,OT_2^1,ETP}\ ((b_0, b_1), \sigma) \equiv do\ \{\\
&\quad (\alpha, \tau) \leftarrow I;\\
&\quad x_\sigma \leftarrow S(\alpha);\\
&\quad y_{1-\sigma} \leftarrow S(\alpha);\\
&\quad let\ y_\sigma = F(\alpha, x_\sigma);\\
&\quad let\ x_\sigma = F^{-1}(\alpha, \tau, y_\sigma);\\
&\quad let\ x_{1-\sigma} = F^{-1}(\alpha, \tau, y_{1-\sigma});\\
&\quad let\ \beta_\sigma = B(\alpha, x_\sigma) \oplus (if\ \sigma\ then\ b_1\ else\ b_0);\\
&\quad let\ \beta_{1-\sigma} = B(\alpha, x_{1-\sigma}) \oplus (if\ \sigma\ then\ b_0\ else\ b_1);\\
&\quad return(\sigma, \alpha, (\beta_\sigma, \beta_{1-\sigma}))\}
\end{aligned}
$$

The only part of the real view we are unable to simulate is the construction of $\beta_{1-\sigma}$ as it requires $b_{1-\sigma}$ — the message party 2 does not receive and thus the simulator does not have access to. However Theorem 3 shows that the following simulator suffices.

$$
\begin{aligned}
&S_{2,OT_2^1,ETP}(\sigma, b_\sigma) \equiv do\ \{\\
&\quad (\alpha, \tau) \leftarrow I;\\
&\quad x_\sigma \leftarrow S(\alpha);\\
&\quad y_{1-\sigma} \leftarrow S(\alpha);\\
&\quad let\ x_{1-\sigma} = F^{-1}(\alpha, \tau, y_{1-\sigma});\\
&\quad let\ \beta_\sigma = B(\alpha, x_\sigma) \oplus b_\sigma;\\
&\quad let\ \beta_{1-\sigma} = B(\alpha, x_{1-\sigma});\\
&\quad return(\sigma, \alpha, (\beta_\sigma, \beta_{1-\sigma}))\}
\end{aligned}
$$

For the case where $b_{1-\sigma} = False$ we have perfect security, the real and ideal views are equal.

Lemma 1. *Assume* $\neg\, b_{1-\sigma}$ *then we have*

$$R_{2,OT_2^1,ETP}((b_0, b_1), \sigma) = ideal_{2,OT_2^1,ETP}((b_0, b_1), \sigma)$$

This implies that $adv\text{-}P_{2,OT_2^1,ETP}((b_0, b_1), \sigma, D) = 0$ (when $b_{1-\sigma} = False$). It is left to consider the case where $b_{1-\sigma} = True$. In this case we construct an adversary, $\mathcal{A}_{HCP}$ that wins the HCP game if D (which is taken as an input) can distinguish the real and simulated views — that is we show a reduction to the HCP assumption.

$$
\begin{aligned}
&\mathcal{A}_{HCP}(\mathcal{A}, \sigma, b_\sigma, D) \equiv do\ \{\\
&\quad \beta_{1-\sigma} \leftarrow coin;\\
&\quad x_\sigma \leftarrow S(\alpha);\\
&\quad let\ \beta_\sigma = B(\alpha, x_\sigma) \oplus b_\sigma;\\
&\quad d \leftarrow D(\sigma, \alpha, \beta_\sigma, \beta_{1-\sigma});\\
&\quad return(if\ d\ then\ \beta_{1-\sigma}\ else\ \neg\beta_{1-\sigma})\}
\end{aligned}
$$

Lemma 2. *Assume* $b_{1-\sigma}$ *then we have*

$$adv\text{-}P_{2,OT_2^1,ETP}((b_0, b_1), \sigma, D) = 2 \cdot adv_{HCP}(\mathcal{A}_{HCP}, \sigma, b_\sigma, D)$$

The proof of Lemma 2 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 however still follows the overall structure of the proof in [24] (the sequence of equalities on p. 14). We find one proof step was formally more difficult to reason about than the others — namely the first equality in the sequence; we are required to split the probability of $\mathcal{A}_{HCP}$ winning the HCP game into two cases, dependent on the coin flip in $\mathcal{A}_{HCP}$ $(\beta_{1-\sigma})$. While splitting the probability based on the outcome of the coin flip is *obvious* in the paper proof, formally we have to work harder. The formal proof 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 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.

Using Lemmas 1 and 2 we bound the advantage for Party 2.

Theorem 3.

$$adv\text{-}P_{2,OT_2^1,ETP}((b_0, b_1), \sigma, D) \leq 2 \cdot adv_{HCP}(\mathcal{A}_{HCP}, \sigma, b_\sigma, D)$$

For Party 1 we are able to construct a simulator, $S_{1,OT_2^1,ETP}$ (shown in Appendix A), in the same manner as in [24] and show it is equal to the real view.

Theorem 4.

$$perfect\text{-}sec\text{-}P_{1,OT_2^1,ETP}((b_0, b_1), \sigma)$$

Together Theorems 3 and 4 show Protocol 1 is secure.

4.4 Instantiating for RSA

It is known that the RSA collection of functions provides an ETP (see [18, Section 2.4.4.2] together with [19, Section C.1]). We formalise this RSA collection and instantiate it for Protocol 1. We fix as a parameter a set of primes (*prime-set* :: *nat set*) that we can sample the parameters for RSA from and define the algorithms that make up the ETP for RSA (I_{RSA}, $domain_{RSA}$, $range_{RSA}$, f_{RSA}). The permutation considered here is,

$$F_{RSA}((N, e), x) = x^e \ mod\ N \tag{13}$$

for appropriately chosen N and e. The other algorithms are given and explained in Appendix B.

To show security for the RSA instantiation of Protocol 1 we use the generality of our work from the previous section and Isabelle's module system. In particular to realise the whole proof of security for the RSA instantiation we only need to prove that assumptions made in the locale *etp-base* are satisfied. The most challenging of these assumptions to prove is that the RSA function (Equation 13) is a permutation.

It is often the case when formalising paper proofs that detailed proofs of *obvious* results are hard to find and while this is a well known result we struggled to find a proof in the literature with sufficient enough detail to be useful in the formalisation.

In this instance, the map's domain and range are equal thus we must show that for any x, y in the domain (or range), if $f(x) = f(y)$ then we have $x = y$. Formally we prove the following.

Lemma 3. *Assume P and Q are primes, $P \neq Q$, e and $(P-1)\cdot(Q-1)$ are coprime, $x, y < P \cdot Q$ and $x^e \ mod(P \cdot Q) = y^e \ mod(P \cdot Q)$ then we have that $x = y$.*

Corollary 1. *Assume $\alpha \in$ set-spmf(I_{RSA}), then we have*

$$bij\text{-}betw(f(\alpha), domain(\alpha), range(\alpha)).$$

This Corollary comes directly from Lemma 3 where the assumptions in Lemma 3 are met due to the construction of I_{RSA}. This is the main proof statement we require to import our proof from the general case to the RSA instantiation.

Theorem 5. *For Party 1 we have perfect security*

$$perfect\text{-}sec\text{-}P_{1,OT_2^1,RSA}((b_0, b_1), \sigma)$$

and for Party 2 we bound the advantage by the HCP advantage

$$adv\text{-}P_{2,OT_2^1,RSA}(((b_0, b_1), \sigma), D) \leq 2 \cdot adv_{HCP}(\mathcal{A}_{HCP}, \sigma, b_\sigma, D).$$

This has shown that, assuming an HCP exists for RSA we can securely compute OT_2^1 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 of the RSA instantiation in full from scratch. However subsequent leveraging of the module system, as outlined above, 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 locale *etp-base* are valid. In fact, no security results need to be proven at all in future instantiations.

4.5 The RSA instantiation in the asymptotic setting

So far all security statements have been considered in the concrete setting, where the security parameter has been assumed to be implicit in all algorithms. Isabelle and CryptHOL provides a method for proving asymptotic security statements by making all definitions and statements dependent on the security parameter allowing us to derive the conventional asymptotic security statements from the original concrete statements. We provide proofs in the asymptotic setting for all protocols we consider — here we present the instantiation of RSA by way of example.

We introduce the security parameter n by replacing the locale parameter *prime-set* by a family of sets of primes *prime-set* :: *nat* $\Rightarrow$ *nat set*. Now the set of primes used in the protocol is dependent on n. After this parameter has been altered to account for n the others become explicitly dependent on it also; for example I_{RSA} samples from *prime-set*, thus it is also now dependent on n.

After importing the concrete setting parametrically for all n; algorithms now depend explicitly on n. Moreover, due to Isabelle's module structure we are able to use results proven in the concrete setting in our newly constructed asymptotic setting. Results from the concrete setting can only be used once it has been proven that the import is valid, something the user is required to do when importing a module. This is similar to importing the general proof of OT_2^1 using HCPs to the RSA instantiation.

We now prove the security results in the asymptotic setting. First we show correctness is still valid and then that security holds.

Theorem 6. *The RSA instantiation of Protocol 1 is correct.*

$$protocol_{OT_2^1,RSA}(n, (b_0, b_1), \sigma) = funct_{OT_2^1}((b_0, b_1), \sigma)$$

Note that the security parameter only appears as inputs to functions where it is used. Equation 5 shows that the security parameter is never required to define $funct_{OT_2^1}$. Security is shown by the following Theorem.

Theorem 7. *For Party 1 we have perfect security, that is,*

$$perfect\text{-}sec\text{-}P_{1,OT_2^1,RSA}(n, (b_0, b_1), \sigma).$$

For Party 2, assume negligible$(\lambda\ n.\ adv_{HCP}(n, \mathcal{A}_{HCP}, b_\sigma, D))$ then we have,

$$negligible(\lambda\ n.\ adv\text{-}P_{2,OT_2^1,RSA}(n, (b_0, b_1), \sigma, D).$$

Thus we have shown the security results in the asymptotic setting.

5 Formalising the GMW protocol

The GMW protocol allows for the secure computation of any boolean circuit. It does so by providing a method for computing gates in the circuit securely. The protocol achieves secure gate computation by using secret sharing among the parties. Intuitively each party splits their input into two parts (shares); keeping one share and sending the other to the other party. The parties work together through the circuit they want to compute, gate by gate. After each gate computation each party holds one share of the output of the gate.

We formalise the security results for computing AND and XOR gates in the two party setting — AND and XOR form a universal set from which we can realise the whole of MPC.

5.1 Secret sharing

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.

$$\begin{array}{l|l} share_{GMW}(x) = do\ \{ & reconstruct_{GMW}(a, b) \\ \quad a \leftarrow coin; & = return(a \oplus b) \\ \quad return(a, x \oplus a)\} & \end{array}$$

Correctness of the sharing scheme comes easily.

Theorem 8. $correct_{share_{GMW}}(x)$.

5.2 Securely computing AND and XOR gates

The GMW protocol provides sub protocols to compute XOR and AND gates on the shared inputs (that have already been shared between the parties). We formalise both sub protocols.

To achieve secure computation of an AND gate we require OT_4^1. We take the protocol that realises OT_k^1 from [19] (Section 7.3.3) and formalise the adapted case for OT_4^1.

Again our proofs here are constructed in a modular way, for the construction of OT_4^1 from OT_2^1 we assume the security results of the underlying OT_2^1. Namely we assume correctness, perfect security for Party 2 and bound the advantage of Party 1. These assumptions correspond to the results obtained by the Noar-Pinkas OT_2^1 [31] which is used in practical implementations of GMW. We leave details of our formalisation of the construction of OT_4^1 from OT_2^1 to Appendix C. Our proofs in this section assume the results from the construction of OT_4^1 from OT_2^1, for clarity we make these assumptions explicit in the security statements we give here.

To compute XOR and AND gates we assume Party 1 has input $x \in \{0, 1\}$ and Party 2 has input $y \in \{0, 1\}$, after sharing their inputs 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 protocol for an XOR gate is as follows.

Protocol 2. *[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 $b_1 \oplus b_2$.*

Correctness comes from the commutativity of the XOR operation. There is no communication between the parties in this protocol and thus security is trivially achieved.

Securely computing an AND gate is more involved. The functionality we want to evaluate is,

$$\begin{array}{l} funct_{AND}((a_1, a_2), (b_1, b_2)) = do\ \{ \\ \quad \sigma \leftarrow coin; \\ \quad return(\sigma, \sigma \oplus (a_1 \oplus b_1) \wedge (a_2 \oplus b_2))\} \end{array}$$

Sampling σ in the functionality results in both outputs being uniformly distributed, failure to do this would mean one party (in this case Party 2) would learn the result of the computation. To realise this functionality we require OT_4^1.

Protocol 3. *[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
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_4^1 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_4^1 while Party 1 keeps σ.*

The protocol is correct as both parties hold a share of the output such that when combined with xor (reconstructed) 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 and the security of the underlying OT_4^1. This results in party 2 receiving one and only one value they can *decrypt* to form their share.

Correctness is proven analogously to the semi honest setting, here we consider security of the protocol.

We refer the reader to the formalisation for details of the proof of security for the XOR gate. Here we show security for the AND gate.

The real and ideal views for the AND gate protocol are given in Appendix D. For party 2 we show perfect security, this comes from the perfect security of the OT_4^1 we use. We can simulate the real view using the simulator for party 2 from the underlying OT_4^1.

Theorem 9. *Assume that we have perfect security for party 2 for the underlying OT_4^1 protocol used then we have.*

$$perfect\text{-}sec\text{-}P_{2,AND}((a_1, a_2), (b_1, b_2)).$$

To show security for Party 1 we show a reduction to the security of OT_4^1.

Theorem 10. *Assume that the advantage for party 1 in the underlying OT_4^1 protocol used is $P_1adv_{OT_4^1}$ then we have,*

$$adv\text{-}P_{1,AND}((a_1, a_2), (b_1, b_2), D) \leq P_1adv_{OT_4^1}.$$

Theorems 9 and 10 show security in the semi-honest model for the AND gate construction given in Protocol 3.

6 Formalising Malicious Security

We now consider the malicious adversary model in the two party setting. First we formalise the definitions of malicious security and then prove an OT_2^1 protocol secure with respect to our definitions.

6.1 Formalising the definitions

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.

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 parties *correct* input to a protocol, a corrupted party may enter the protocol with an input that is not the original 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 [19, Section 7.2.3].

A protocol is said to be secure if the adversary's behaviour is limited to these three actions. We consider the malicious security definitions from [19, Section 7.2.3] and [22, Section 2.3.1] where an ideal and real world are considered.

The ideal model uses a trusted party that provides 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[3] available to the adversary. The ideal model is constructed as follows [22]:

- **Send inputs to trusted party:** The honest party sends its 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 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.

The output of the ideal model, when Party i is corrupted, is denoted as $IDEAL_{f,\mathcal{A}(z),i}(x, y)$ — the output of both parties in the ideal model. The output of the real model ($REAL_{\pi,\mathcal{A}(z),i}(x, y)$) is the output of each party in a real execution of the protocol where all messages for the corrupted party, i, are sent by the adversary. Informally, a protocol π securely computes f with abort in the presence of malicious adversaries if for all $\mathcal{A}$ in the real model there exists a simulator S that interacts with the ideal model such that the $IDEAL_{f,S(z),i}(x, y)$ and $REAL_{\pi,\mathcal{A}(z),i}(x, y)$ are indistinguishable for $i \in \{1, 2\}$.

To make the definitions of malicious security we, as usual, construct a locale (*malicous-base*) and fix the parameters we require to 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 celar 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 we construct some type synonyms for readability, in particular we consider the types of the simulators.

We show the types for Party 1, the types for Party 2 are analogous. The simulator interacts with the idea model and has two components. We define their types separately and then combine them. The first component of the simulator is used in the first phase of the ideal model, that is it sends the adversaries chosen input to the trusted party. It takes as input the real world adversary, 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. It also outputs its state, that can be passed to the second component.

$$\textit{type-syn}\ (`\mathcal{A}^{P_1}_{real}, `in_1, `aux, `s_1)\ S_1^{P_1} = \\ `\mathcal{A}^{P_1}_{real} \Rightarrow `in_1 \Rightarrow `aux \Rightarrow (`in \times `s_1)\ spmf$$

The second component of the simulator for Party 1 outputs the corrupted parties output. It is allowed to also see the output of the protocol, as given to it by the trusted party in the ideal game as well as the state outputted by the first component of the adversary. This part of the simulator outputs whatever it likes — we represent this by allowing it to output something of abstract type.

$$\textit{type-syn}\ (`\mathcal{A}^{P_1}_{real}, `in_1, `aux, `out_1, `s_1, `\mathcal{A}_{out_1})\ S_2^{P_1} = \\ `\mathcal{A}^{P_1}_{real} \Rightarrow `in_1 \Rightarrow `aux \Rightarrow `out_1 \Rightarrow `s_1 \Rightarrow `\mathcal{A}_{out_1}\ spmf$$

We combine the two components of the simulator as a tuple.

$$\textit{type-syn}\ (`\mathcal{A}^{P_1}_{real}, `in_1, `aux, `out_1, `s_1, `\mathcal{A}_{out_1})\ S^{P_1} = \\ (`\mathcal{A}_{real,P_1}, `in_1, `aux, `s_1)\ S_1^{P_1} \times \\ (`\mathcal{A}^{P_1}_{real}, `in_1, `aux, `out_1, `s_1, `\mathcal{A}_{out_1})\ S_2^{P_1}$$

The locale we construct to make our malicious definitions is as follows.

locale *malicious-base* =
  fixes *funct* :: $`in_1 \Rightarrow `in_2 \Rightarrow (`out_1 \times `out_2)\ spmf$
  and *protocol* :: $`in_1 \Rightarrow `in_2 \Rightarrow (`out_1 \times `out_2)\ spmf$
  and S_1^1 :: $(`\mathcal{A}^{P_1}_{real}, `in_1, `aux, `s_1)\ S_1^{P_1}$
  and S_1^2 :: $(`\mathcal{A}^{P_1}_{real}, `in_1, `aux, `out_1, `s_1, `\mathcal{A}_{out_1})\ S_2^{P_1}$
  and R_1 :: $`in_1 \Rightarrow `in_2 \Rightarrow `aux \Rightarrow `\mathcal{A}^{P_1}_{real} \Rightarrow (`\mathcal{A}_{out_1} \times `out_2)\ spmf$
  and S_2^1 :: $(`\mathcal{A}^{P_2}_{real}, `in_2, `aux, `s_2)\ S_{1,P_2}$
  and S_2^2 :: $(`\mathcal{A}^{P_2}_{real}, `in_2, `aux, `out_2, `s_2, `\mathcal{A}_{out_2})\ S_2^{P_2}$
  and R_2 :: $`in_1 \Rightarrow `in_2 \Rightarrow `aux \Rightarrow `\mathcal{A}^{P_2}_{real} \Rightarrow (`out_1 \times `\mathcal{A}_{out_2})\ spmf$

[3] This is a priori information.

In the same way as the semi-honest setting, for a protocol to be correct we require *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.

To consider security we define the ideal model. For clarity we define the trusted party as the functionality,

$$\textit{trusted-party}(x, y) = \textit{funct}(x, y)$$

, this is the action the trusted party performs.

Our formalisation of the ideal model for Party 1 is defined by the probabilistic program *ideal-model*$_1$.

$$\begin{aligned}
&\textit{ideal-model}_1(x, y, z, \mathcal{A}) = \textit{do}\ \{\\
&\quad \textit{let}\ (\mathcal{A}_1, \mathcal{A}_2) = \mathcal{A};\\
&\quad (x', \textit{aux}_{out}) \leftarrow \mathcal{A}_1(x, z);\\
&\quad (\textit{out}_1, \textit{out}_2) \leftarrow \textit{trusted-party}(x', y);\\
&\quad \textit{out}'_1 \leftarrow \mathcal{A}_2(x', z, \textit{out}_2, \textit{aux}_{out});\\
&\quad \textit{return}(\textit{out}'_1, \textit{out}_2)\}
\end{aligned}$$

We make two remarks about this definition; the first concerns aborts and the second the state of the adversary.

1. We do not explicitly model the abort procedures as they are covered by the type *spmf* — the adversary can output nothing and thus terminate the program.
2. As we split the adversary into two parts it must be allowed to pass state. We incorporate the state into $\textit{aux}_{out}$ in the instantiated proofs.

The ideal view in the malicious setting is defined as the output of the ideal model when the simulator send messages on behalf of the adversary.

Definition 6. *We define the ideal view of Party 1 as,*

$$\begin{aligned}
&\textit{mal}_{\textit{ideal}_1}(x, y, z, \mathcal{A})\\
&\qquad \equiv \textit{ideal-model}_1(x, y, z, (S^1_1(\mathcal{A}), S^2_1(\mathcal{A}))).
\end{aligned}$$

$\mathcal{A}$ consists of a tuple of algorithms, one 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 consider the advantage a distinguisher has of distinguishing them. For perfect security we require equality between the views.

Definition 7 (Perfect security for Party 1)**.**

$$\begin{aligned}
&\textit{perfect-sec-P}_1(x, y, z, \mathcal{A})\\
&\qquad = (\textit{mal}_{\textit{real}_1}(x, y, z, \mathcal{A}) = \textit{mal}_{\textit{ideal}_1}(x, y, z, \mathcal{A}))
\end{aligned}$$

The advantage of a distinguisher is defined as follows.

Definition 8 (Advantage: Party 1)**.**

$$\begin{aligned}
&\textit{mal}_{\textit{adv}_1}(x, y, z, \mathcal{A}, D) =\\
&\qquad |\mathcal{P}[(\textit{mal}_{\textit{real}_1}(x, y, z, \mathcal{A}) \rhd D) = \textit{True}] -\\
&\qquad\qquad \mathcal{P}[(\textit{mal}_{\textit{ideal}_1}(x, y, z, \mathcal{A}) \rhd D) = \textit{True}]|
\end{aligned}$$

The work of Haagh et al. [21] formalises the same malicious model (active security model) in EasyCrypt, however as discussed in Section 1.2 a meta (paper) theorem was required to link the formalisation to the paper definitions.

6.2 A protocol realising OT^1_2 in the malicious setting

In this section we show the definitions we provide for malicious security are satisfied by the OT^1_2 protocol from [22, 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 Diffie Helman (DH) relation (F^{DH}_{ZK}). The DH relation for a group G with generator g is a tuple $((h, a, b), r)$ where $a = g^r$ and $b = h^r$. Specifically, the functionality is as follows,

$$\begin{aligned}
&F^{DH}_{ZK}((h, a, b), ((h', a', b'), r)) =\\
&\quad \textit{return}(a = g^r \wedge b = h^r \wedge (h, a, b) = (h', a', b'), _). \qquad (14)
\end{aligned}$$

In the context of Protocol 4, 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. Party 1 also learns if the input made by Party 2 is the same as their input.

The protocol uses a cyclic group G, with generator g, and where the Decisional Diffie Helman (DDH) assumption holds. The DDH assumption informally states that the tuples $(g^a, g^b, g^{a \cdot b})$ and (g^a, g^b, g^c), where $a, b, c \xleftarrow{\$} \mathbb{Z}_{|G|}$, are indistinguishable. It was formalised as part of CryptHOL.

The OT^1_2 protocol we consider is given below in Protocol 4

Protocol 4. *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* $\alpha_0, \alpha_1, r \leftarrow \{1, ..., |G|\}$ *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* $u_0, v_0, u_1, v_1 \leftarrow \{1, ..., |G|\}$, *and computes* (e_0, e_1) *and sends the tuple to R:*
$e_0 = (w_0, z_0)$ *where* $w_0 = a^{u_0} \cdot g^{v_0}$, $z_0 = b_0^{u_0} \cdot h_0^{v_0} \cdot m_0$.
$e_1 = (w_1, z_1)$ *where* $w_1 = a^{u_1} \cdot g^{v_1}$, $z_1 = (\frac{b_1}{g})^{u_1} \cdot h_1^{v_1} \cdot m_1$.

6. *R outputs* $\frac{z_\sigma}{w_\sigma^{\alpha_\sigma}}$ *and S outputs nothing.*

Security for the Receiver is upheld because σ is sent only as an exponent of the generator which is masked by random group element. 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 thus security for the Sender is achieved.

6.3 Proving OT_2^1 secure in the malicious setting

In this section we discuss our formalisation of security proof of Protocol 4. First we show correctness of the protocol.

Theorem 11. *Assume* $m_0, m_1 \in G$ *then,*

$$funct_{OT_2^1}((m_0, m_1), \sigma) = protocol_{OT_2^1, mal}((m_0, m_1), \sigma).$$

Here $protocol_{OT_2^1, mal}$ is the probabilistic program that defines the output of the protocol defined in Protocol 4. Isabelle had to be helped more extensively in the rewriting of terms here compared to other proofs of correctness. This was due to the more complex constructions in the protocol.

To prove security of Protocol 4 we 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.

6.3.1 Variant of the DDH assumption

Traditionally the DDH assumption states that the tuples (g^x, g^y, g^z) and (g^x, g^y, g^{xy}) are hard to distinguish, the variant we consider 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 at least as hard as the original DDH assumption.

Lemma 4.

$$DDH\text{-}adv_{var}(\mathcal{A}) \leq DDH\text{-}adv(\mathcal{A}) + DDH\text{-}adv(\mathcal{A}'(\mathcal{A}))$$

Where $DDH\text{-}adv$ is the original DDH advantage (formalised in [29]), $DDH\text{-}adv_{var}$ 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.

6.3.2 Party 1

The simulators used to show security are as follows:

$$\begin{aligned}
&S_{1,P_1}((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), M, z) = do\ \{\\
&\quad r, \alpha_0, \alpha_1 \leftarrow uniform(|G|);\\
&\quad let\ h_0 = g^{\alpha_0};\\
&\quad let\ h_1 = g^{\alpha_1};\\
&\quad let\ a = g^r;\\
&\quad let\ b_0 = h_0^r;\\
&\quad let\ b_1 = h_1^r \otimes g;\\
&\quad ((in_1, in_2, in_3), s) \leftarrow \mathcal{A}_1(M, h_0, h_1, a, b_0, b_1, z);\\
&\quad let\ (h, a, b) = (h_0 \otimes h_1^{-1}, a, b_0 \otimes b_1^{-1});\\
&\quad _ \leftarrow assert((in_1, in_2, in_3) = (h, a, b));\\
&\quad (((w_0, z_0), (w_1, z_1)), s') \leftarrow \mathcal{A}_2(h_0, h_1, a, b_0, b_1, M, s);\\
&\quad let\ x_0 = z_0 \otimes w_0^{-\alpha_0};\\
&\quad let\ x_1 = z_1 \otimes w_1^{-\alpha_1};\\
&\quad return((x_0, x_1), s')\}
\end{aligned}$$

$$S_{2,P_1}((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), M, z, out_1, s') = \mathcal{A}_3(s')$$

The only difference between the simulator and what happens in a real execution of the protocol is that the values b_0 and b_1 are incorrectly generated. To show security for Party 1 we make a case split on σ and construct an adversary for each case ($DDH\text{-}\mathcal{A}_{\sigma=1}$, $DDH\text{-}\mathcal{A}_{\sigma=0}$[4]) that *break* the variant of the DDH assumption and then use Lemma 4 to bound the advantages by the traditional DDH assumption advantage. We construct the DDH adversary for the case $\sigma = 0$ as follows.

$$\begin{aligned}
&DDH\text{-}\mathcal{A}_{\sigma=0}(M, z, (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), D, h, a, t) = do\ \{\\
&\quad \alpha_0 \leftarrow uniform(|G|);\\
&\quad let\ h_0 = g^{\alpha_0};\\
&\quad let\ h_1 = h;\\
&\quad let\ b_0 = a^{\alpha_0};\\
&\quad let\ b_1 = t;\\
&\quad ((in_1, in_2, in_3), s) \leftarrow \mathcal{A}_1(M, h_0, h_1, a, b_0, b_1, z);\\
&\quad _ \leftarrow assert(in_1 = h_0 \otimes h_1^{-1} \wedge in_2 = a \wedge in_3 = b_0 \otimes b_1^{-1});\\
&\quad (((w_0, z_0), (w_1, z_1)), s') \leftarrow \mathcal{A}_2(h_0, h_1, a, b_0, b_1, M, s);\\
&\quad let\ x_0 = z_0 \otimes w_0^{-\alpha_0}\\
&\quad adv_{out} \leftarrow \mathcal{A}_3(s');\\
&\quad D(adv_{out}, x_0)\}
\end{aligned}$$

The intuition is that when the last three inputs, (h, a, t), are a DDH tuple the adversary acts as though it is in the real model and when the inputs are not a DDH tuple it follows the ideal model.

Theorem 12. *In the case where* $\sigma = 0$ *then we have*

$$\begin{aligned}
&mal_{adv_1}((m_0, m_1), \sigma, z, S_1, \mathcal{A}, D)) \leq\\
&\quad DDH\text{-}adv(DDH\text{-}\mathcal{A}_{\sigma=0}((m_0, m_1), z, \mathcal{A}, D)) +\\
&\quad\quad DDH\text{-}adv(\mathcal{A}'(DDH\text{-}\mathcal{A}_{\sigma=0}((m_0, m_1), z, \mathcal{A}, D)))
\end{aligned}$$

[4]Here we use 1 as an encoding for *True* and 0 as an encoding for *False*. The adversary $DDH\text{-}\mathcal{A}_{\sigma=1}$, is given in Appendix E along with some intuition about its construction.

In the case where $\sigma = 1$ then we have,

$$mal_{adv_1}((m_0, m_1), \sigma, z, S_1, \mathcal{A}, D)) \leq$$
$$DDH\text{-}adv(DDH\text{-}\mathcal{A}_{\sigma=1}((m_0, m_1), z, \mathcal{A}, D)) +$$
$$DDH\text{-}adv(\mathcal{A}'(DDH\text{-}\mathcal{A}_{\sigma=1}((m_0, m_1), z, \mathcal{A}, D)))$$

6.3.3 Party 2

To show security for Party 2 we construct the simulator, $S_2 = (S_{1,P_2}, S_{2,P_2})$ and show that the real and ideal model are equal.

$$\begin{aligned}
&S_{1,P_2}((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), \sigma, z) = do\ \{\\
&\quad (h_0, h_1, a, b_0, b_1) \leftarrow \mathcal{A}_1(\sigma);\\
&\quad \text{-} \leftarrow assert(h_0, h_1, a, b_0, b_1 \in G);\\
&\quad ((in_1, in_2, in_3), r) \leftarrow \mathcal{A}_2(h_0, h_1, a, b_0, b_1);\\
&\quad let\ (h, a, b) = (\tfrac{h_0}{h_1}, a, \tfrac{b_0}{b_1});\\
&\quad b, \text{-}) \leftarrow F^{DH}_{ZK}((h, a, b), ((in_1, in_2, in_3), r));\\
&\quad \text{-} \leftarrow assert(b);\\
&\quad let\ l = \tfrac{b_0}{h_0^r};\\
&\quad return((if\ l = 1\ then\ False\ else\ True), (h_0, h_1, a, b_0, b_1))\}
\end{aligned}$$

$$\begin{aligned}
&S_{2,P_2}((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), \sigma', z, m_\sigma, aux_{out}) = do\ \{\\
&\quad let\ (h_0, h_1, a, b_0, b_1) = aux_{out};\\
&\quad u_0, v_0, u_1, v_1 \leftarrow uniform(|G|);\\
&\quad ((in_1, in_2, in_3), r) \leftarrow \mathcal{A}_2(h_0, h_1, a, b_0, b_1);\\
&\quad let\ w_0 = a^{u_0} \otimes g^{v_0};\\
&\quad let\ w_1 = a^{u_1} \otimes g^{v_1};\\
&\quad let\ z_0 = b_0^{u_0} \otimes h_0^{v_0} \otimes (if\ \sigma'\ then\ 1\ else\ m_\sigma);\\
&\quad let\ z_1 = (\tfrac{b_1}{g})^{u_1} \otimes h_1^{v_1} \otimes (if\ \sigma'\ then\ m_\sigma\ else\ 1);\\
&\quad \mathcal{A}_3((w_0, z_0), (w_1, z_1))\}
\end{aligned}$$

To show equality between the real and ideal views we consider the cases on $l = \frac{b_0}{h_0^r}$: $l = 1$, $l = g$, $l \notin \{1, g\}$. Like the case split in the proof of Theorem 3 we must reason about a bound variable within a probabilistic program and thus the proof of Theorem 13 is involved and technical.

Theorem 13. *Assume $m_0, m_1 \in G$ then we have*

$$perfect\text{-}sec\text{-}P_2((m_0, m_1), \sigma, z, \mathcal{A}).$$

In proving Theorem 13 we were able to closely follow the paper proof from [22].

7 Conclusion and future work

In this paper we have formalised both the semi-honest and malicious models and instantiated *OT* protocols in both as well as the two party GMW protocol in the semi-honest setting. Our work has shown that CryptHOL is a suitable and usable framework for cryptographic proofs. In fact in our work we only use parts of CryptHOLs rich formalisation, in particular we do not require any of the machinery for constructing complex adversaries that is used for example in [28]. We feel this is a benefit as have shown that much can be achieved with CryptHOL without the need to use many of the extremely technical parts of the tool.

Achieving a proof of security for malicious GMW would require a large proof effort and is left as future work. Significant extensions towards a formalisation of Zero Knowledge would need to be made to [12, 14] as well as extending this work to the *n* party setting.

Acknowledgements We would like to thank the reviewers for their detailed and helpful reviews as well as Andreas Lochbihler for his continuous support and development of CryptHOL. This work was supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1.

References

[1] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, and Vitor Pereira. 2017. A Fast and Verified Software Stack for Secure Function Evaluation. In *ACM Conference on Computer and Communications Security*. ACM, 1989–2006.

[2] David Aspinall and David Butler. 2019. Multi-Party Computation. *Archive of Formal Proofs* 2019 (2019).

[3] G Barthe, B Grégoire, and S Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. In *POPL*. ACM, 90–101.

[4] G Barthe, B Grégoire, S Heraud, and S Zanella Béguelin. 2011. Computer-Aided Security Proofs for the Working Cryptographer. In *CRYPTO (Lecture Notes in Computer Science)*, Vol. 6841. Springer, 71–90.

[5] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. 2017. CryptHOL: Game-based Proofs in Higher-order Logic. *IACR Cryptology ePrint Archive* 2017 (2017), 753.

[6] Mihir Bellare and Phillip Rogaway. 2006. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In *EUROCRYPT (Lecture Notes in Computer Science)*, Vol. 4004. Springer, 409–426.

[7] Bruno Blanchet. 2008. A Computationally Sound Mechanized Prover for Security Protocols. *IEEE Trans. Dependable Sec. Comput.* 5, 4 (2008), 193–207.

[8] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *ESORICS (Lecture Notes in Computer Science)*, Vol. 5283. Springer, 192–206.

[9] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. 2009. Secure Multiparty Computation Goes Live. In *Financial Cryptography (Lecture Notes in Computer Science)*, Vol. 5628. Springer, 325–343.

[10] David Butler and David Aspinall. 2019. Multi Party Computation. *Archive of Formal Proofs* (2019). https://www.isa-afp.org/entries/Multi_Party_Computation.html, Formal proof development.

[11] 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.

[12] 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.

[13] David Butler and Andreas Lochbihler. 2019. Sigma Protocols and Commitment Schemes. *Archive of Formal Proofs* (2019). https://www.isa-afp.org/entries/Sigma_Commit_Crypto.html, Formal proof development.

[14] David Butler, Andreas Lochbihler, David Aspinall, and Adrià Gascón. 2019. Formalising Σ-Protocols and Commitment Schemes using CryptHOL. *IACR Cryptology ePrint Archive* 2019 (2019), 1185.
[15] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*. IEEE Computer Society, 136–145.
[16] Ran Canetti, Alley Stoughton, and Mayank Varia. 2019. EasyUC: Using EasyCrypt to Mechanize Proofs of Universally Composable Security. In *Proceedings of the 32nd IEEE Computer Security Foundations Symposium (CSF 2019)*. IEEE Computer Society, Hoboken, NJ, USA.
[17] Shimon Even, Oded Goldreich, and Abraham Lempel. 1985. A Randomized Protocol for Signing Contracts. *Commun. ACM* 28, 6 (1985), 637–647.
[18] Oded Goldreich. 2001. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press.
[19] Oded Goldreich. 2004. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press.
[20] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*. ACM, 218–229.
[21] Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. 2018. Computer-Aided Proofs for Multiparty Computation with Active Security. In *CSF*. IEEE Computer Society, 119–131.
[22] Carmit Hazay and Yehuda Lindell. 2010. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Springer.
[23] John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. 2014. Application-Scale Secure Multiparty Computation. In *ESOP (Lecture Notes in Computer Science)*, Vol. 8410. Springer, 8–26.
[24] Yehuda Lindell. 2017. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*. Springer International Publishing, 277–346.
[25] Andreas Lochbihler. [n. d.]. CryptHOL. *Archive of Formal Proofs* 2017 ([n. d.]).
[26] Andreas Lochbihler. 2016. Probabilistic Functions and Cryptographic Oracles in Higher Order Logic. In *ESOP (Lecture Notes in Computer Science)*, Vol. 9632. Springer, 503–531.
[27] Andreas Lochbihler and S. Reza Sefidgar. [n. d.]. Constructive Cryptography in HOL. *Archive of Formal Proofs* 2018 ([n. d.]).
[28] Andreas Lochbihler, S. Reza Sefidgar, David A. Basin, and Ueli Maurer. 2019. Formalizing Constructive Cryptography using CryptHOL. In *Computer Security Foundations (CSF 2019)*. IEEE, 152–166.
[29] Andreas Lochbihler, S. Reza Sefidgar, and Bhargav Bhatt. [n. d.]. Game-based cryptography in HOL. *Archive of Formal Proofs* 2017 ([n. d.]).
[30] Ueli M. Maurer. 2006. Secure multi-party computation made simple. *Discrete Applied Mathematics* 154, 2 (2006), 370–381.
[31] Moni Naor and Benny Pinkas. 2001. Efficient oblivious transfer protocols. In *SODA*. ACM/SIAM, 448–457.
[32] A Petcher and G Morrisett. 2015. The Foundational Cryptography Framework. In *POST (Lecture Notes in Computer Science)*, Vol. 9036. Springer, 53–72.
[33] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613.
[34] Victor Shoup. 2004. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive* 2004 (2004), 332.
[35] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *FOCS*. IEEE Computer Society, 160–164.
[36] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *FOCS*. IEEE Computer Society, 162–167.

A Simulator for Party 1 in the Protocol 1

The simulator used to show perfect security in Theorem 4 is as follows.

$$
\begin{aligned}
&S_{1,OT_2^1,ETP}((b_0, b_1), \_) = do\ \{\\
&\quad (\alpha, \tau) \leftarrow I;\\
&\quad y_0 \leftarrow S(\alpha)\\
&\quad y_1 \leftarrow S(\alpha)\\
&\quad return((b_0, b_1), y_0, y_1)\}
\end{aligned}
$$

Here the second input to the simulator is empty (the unit type) as there is no output for party 1 from the functionality.

B RSA instantiation

Section 4.4 showed how we instantiated the general OT_2^1 construction using ETPs for the RSA collection of functions. Here we provide more detail on this part of the formalisation, in particular how we formalise the RSA ETP.

To formalise this instantiation we first fix the locale *rsa-base*.

$$
\begin{aligned}
&\textbf{locale}\ \textit{rsa-base} =\\
&\quad \textbf{fixes}\ \textit{prime-set}\ :: \textit{nat set}\\
&\qquad \textbf{and}\ B\ :: \text{`}\textit{index} \Rightarrow \textit{nat} \Rightarrow \textit{bool}\\
&\quad \textbf{assumes}\ \textit{prime-set} \subseteq \{x.\ \textit{prime}(x) \wedge x > 2\}\\
&\qquad \textbf{and}\ \textit{finite}(\textit{prime-set})\\
&\qquad \textbf{and}\ \textit{card}(\textit{prime-set}) > 2
\end{aligned}
$$

The parameter *prime-set* is the set of primes that the parameters of the RSA function are sampled from and B is the HCP we fix — recall we must assume that this exists. The assumptions we make ensure that the set of primes has the desired properties. To define the algorithms for the RSA ETP we first define three simple sampling algorithms we will use.

$$\textit{sample-primes} = \textit{samp-uniform}(\textit{prime-set})$$

$$\textit{sample-set-excl}(Q, P) = \textit{samp-uniform}(Q - P)$$

$$\textit{sample-corpime} = \textit{samp-uniform}(\textit{coprime-set})$$

where $\textit{coprime-set}(N) = \{x.\ \textit{coprime}(x, N) \wedge x > 1 \wedge x < N\}$ is the set of all natural numbers coprime to N in the desired range.

Using these we define I_{RSA} that samples the index and trapdoor as follows.

$$
\begin{aligned}
&I_{RSA} = do\ \{\\
&\quad P \leftarrow \textit{sample-primes};\\
&\quad Q \leftarrow \textit{sample-set-excl}(\textit{prime-set}, P);\\
&\quad let\ N = P \cdot Q;\\
&\quad let\ N' = (P - 1) \cdot (Q - 1);\\
&\quad e \leftarrow \textit{sample-corpime}(N');\\
&\quad let\ d = inv_N(e);\\
&\quad return((N, e), d)\}
\end{aligned}
$$

Here the index is the tuple (N, e) and the trapdoor is d where d is the multiplicative inverse of e modulo $(P - 1) \cdot (Q - 1)$.

It is left to define the domain and range of the RSA collection,

$$domain(N, e) = \{0, \ldots, N\}$$
$$range(N, e) = \{0, \ldots, N\}.$$

Using these algorithms we show that the RSA collection is an instance of the *etp-base* locale and can realise the proof of security in Section 4.4.

C OT_4^1 from OT_2^1

To realise the secure computation of the AND gate in the GMW protocol we require OT_4^1. Here we show how OT_4^1 can be constructed from OT_2^1. In particular, we show how we use the modular structure in Isabelle to assume the required results on OT_2^1 to show security of a protocol that realises OT_4^1.

C.1 A protocol that realises OT_4^1

We take the protocol that realises OT_k^1 from [19, Section 7.3.3] but adapt it for the case of OT_4^1. The functionality for OT_4^1 is defined as,

$$funct_{OT_4^1}((b_{0,0}, b_{0,1}, b_{1,0}, b_{1,1}), (c_0, c_1)) = return((), b_{c_0,c_1}).$$

The protocol is run between the sender S and the receiver R.

Protocol 5. *S has input $(b_{0,0}, b_{0,1}, b_{1,0}, b_{1,1}) \in \{0,1\}^4$ and R has input $(c_0, c_1) \in \{0,1\}^2$.*

1. *S uniformly randomly samples $S_i \leftarrow \{0,1\}$ for $i \in \{0, \ldots, 5\}$.*
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_2^1 protocols together. That is they run,*
   $(\text{-}, S_i) \leftarrow OT_2^1((S_0, S_1), c_0)$
   $(\text{-}, S_j) \leftarrow OT_2^1((S_2, S_3), c_1)$
   $(\text{-}, S_k) \leftarrow OT_2^1((S_4, S_5), c_1)$
4. *R calculates b_{c_0,c_1} where*

$$b_{c_0,c_1} = S_i \oplus (\textit{if } c_0 \textit{ then } S_k \textit{ else } S_j) \oplus (\textit{if } c_0 \textit{ then } (\textit{if } c_1 \textit{ then } \alpha_3 \textit{ else } \alpha_2) \textit{ else } (\textit{if } c_1 \textit{ then } \alpha_1 \textit{ else } \alpha_0)).$$

Correctness of the protocol comes from the assumption of correctness of the OT_2^1. Security comes from the masking of the messages sent from S to R in Step 2 and the security of the OT_2^1.

C.2 Formalising the protocol and its security

To prove security of Protocol 5 we follow a similar procedure as outlined in Section 4. Here we focus on the modularity of the proof we introduce.

As in paper proofs of protocols of this kind — where one uses the underlying security of another protocol — we would like to reuse previous security theorems rather than construct every proof from scratch. In particular, here we want to use the security results from OT_2^1. To achieve this we make the assumptions on the security of OT_2^1 in the locale OT_4^1*-base*.

locale OT_4^1*-base* =
  fixes $protocol_{OT_2^1}$ :: $(bool \times bool) \Rightarrow bool \Rightarrow (unit \times bool)$ *spmf*
    and R_{1,OT_2^1} :: $(bool \times bool) \Rightarrow bool \Rightarrow$ ‘$view_{1,OT_2^1}$ *spmf*
    and S_{1,OT_2^1} :: $(bool \times bool) \Rightarrow unit \Rightarrow$ ‘$view_{1,OT_2^1}$ *spmf*
    and R_{2,OT_2^1} :: $(bool \times bool) \Rightarrow bool \Rightarrow$ ‘$view_{2,OT_2^1}$ *spmf*
    and S_{2,OT_2^1} :: $bool \Rightarrow bool \Rightarrow$ ‘$view_{2,OT_2^1}$ *spmf*
    and $P_1adv_{OT_2^1}$ :: *real*
  assumes $protocol_{OT_2^1}((m_0, m_1), \sigma) = funct_{OT_2^1}((m_0, m_1), \sigma)$
    and $adv\text{-}P_{1,OT_2^1}((m_0, m_1), \sigma, D) \leq P_1adv_{OT_2^1}$
    and $perfect\text{-}sec\text{-}P_{2,OT_2^1}((m_0, m_1), \sigma)$

On the underlying OT_2^1, we assume: correctness, perfect security for Party 2 and bound the advantage of Party 1. These are the security results of the Noar Pinkas OT_2^1 which is used in practical implementations of GMW.

Correctness of the construction given in Protocol 5 comes from the assumed correctness of OT_2^1.

Theorem 14.

$$protocol_{OT_4^1}(B, C) = funct_{OT_4^1}(B, C)$$

The probabilistic program $protocol_{OT_4^1}$ provides the output distribution of Protocol 5 and B and C are the inputs for Party 1 and Party 2 respectively.

To prove security of Party 1 we show a reduction to the security for Party 1 of the underlying OT_2^1. Protocol 5 calls the OT_2^1 protocol three times whereas the simulator can call the simulator the OT_2^1 protocol.

$R_{1,OT_4^1}(B, (c_0, c_1)) = do$ {
$S_0, S_1, S_2, S_3, S_4, S_5 \leftarrow coin$;
$a \leftarrow R_{1,OT_2^1}((S_0, S_1), c_0)$;
$b \leftarrow R_{1,OT_2^1}((S_2, S_3), c_1)$;
$c \leftarrow R_{1,OT_2^1}((S_4, S_5), c_1)$;
$return(B, (S_0, S_1, S_2, S_3, S_4, S_5), a, b, c)$}

$S_{1,OT_4^1}(B, \_) = do$ {
$S_0, S_1, S_2, S_3, S_4, S_5 \leftarrow coin$;
$a \leftarrow S_{1,OT_2^1}((S_0, S_1), \_)$;
$b \leftarrow S_{1,OT_2^1}((S_2, S_3), \_)$;
$c \leftarrow S_{1,OT_2^1}((S_2, S_3), \_)$;
$return(B, (S_0, S_1, S_2, S_3, S_4, S_5), a, b, c)$}

Using this simulator we show the following reduction.

Theorem 15. *Under the assumptions given in the locale OT_4^1-base we have security for the sender in Protocol 5.*

$$adv\text{-}P_{1,OT_4^1}(B, C, D) \leq 3 \cdot P_1adv_{OT_2^1}.$$

A paper proof would likely state that the reduction holds because Protocol 5 uses three calls to the OT_2^1 protocol.

Clearly we must work harder. We prove a distinguisher cannot distinguish between the real and simulated views for Party 1 in the Protocol 5 with greater advantage than $3 \cdot P_1 adv_{OT_2^1}$ 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_2^1} three times whereas the simulated view calls S_{1,OT_2^1} 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. The first intermediate view changes the first call of R_{1,OT_2^1} in the real view to S_{1,OT_2^1}, the second further changes the second call of R_{1,OT_2^1} to S_{1,OT_2^1}. We informally depict this in the diagram below:

$$R_{1,OT_2^1} \prec_{P_1 adv_{OT_2^1}} inter_{view_1} \prec_{P_1 adv_{OT_2^1}} inter_{view_2} \prec_{P_1 adv_{OT_2^1}} S_{1,OT_2^1}$$

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 $3 \cdot P_1 adv_{OT_2^1}$. In this case Theorem 15 becomes,

$$R_{1,OT_2^1} \prec_{3 \cdot P_1 adv_{OT_2^1}} S_{1,OT_2^1}.$$

To prove security for Party 2 we directly use the perfect security result we assume for Party 2 in the OT_2^1.

Theorem 16. *Under the assumptions given in the locale OT_4^1-base we have perfect security for the receiver in Protocol 5. That is we can construct a simulator S_{2,OT_4^1} such that*

$$\textit{perfect-sec-P}_{2,OT_4^1}(B, C)$$

It was semi-technical to use the assumed result on OT_2^1 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 $\textit{perfect-sec-P}_{2,OT_2^1}$) — again the challenge is that this input is embedded within the probabilistic program.

Together Theorems 15 and 16 have shown security for Protocol 5 with respect to the assumptions on the locale OT_4^1-base.

D Security for AND gate

To simulate the real views in the AND gate protocol given in Protocol 3 we call the simulator instead of the real view from the underlying OT_4^1. The real and simulated views are given below for both parties.

$\textit{real-view}_{1,AND}((a_1, a_2), (b_1, b_2)) = do \{$
$\quad \sigma \leftarrow coin;$
$\quad let\ s_0 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus False);$
$\quad let\ s_1 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus True);$
$\quad let\ s_2 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus False);$
$\quad let\ s_3 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus True);$
$\quad V \leftarrow R_{1,OT_4^1}((s_0, s_1, s_2, s_3), (b_1, b_2));$
$\quad (\_, s) \leftarrow protocol_{OT_4^1}((s_0, s_1, s_2, s_3), (b_1, b_2);$
$\quad return(((a_1, a_2), \sigma, V), (\sigma, s))\}$

$\textit{ideal-view}_{1,AND}((a_1, a_2), (b_1, b_2), \sigma) = do \{$
$\quad let\ s_0 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus False);$
$\quad let\ s_1 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus True);$
$\quad let\ s_2 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus False);$
$\quad let\ s_3 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus True);$
$\quad V \leftarrow S_{1,OT_4^1}((s_0, s_1, s_2, s_3), ());$
$\quad return(((a_1, a_2), \sigma, V), (\sigma, \sigma \oplus ((a_1 \oplus b_1) \wedge (a_2 \oplus b_2))))\}$

$\textit{real-view}_{2,AND}((a_1, a_2), (b_1, b_2)) = do \{$
$\quad \sigma \leftarrow coin;$
$\quad let\ s_0 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus False);$
$\quad let\ s_1 = \sigma \oplus (a_1 \oplus False) \wedge (b_1 \oplus True);$
$\quad let\ s_2 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus False);$
$\quad let\ s_3 = \sigma \oplus (a_1 \oplus True) \wedge (b_1 \oplus True);$
$\quad V \leftarrow R_{2,OT_4^1}((s_0, s_1, s_2, s_3), (b_1, b_2));$
$\quad (\_, out_2) \leftarrow protocol_{OT_4^1}((s_0, s_1, s_2, s_3), (b_1, b_2);$
$\quad return(((b_1, b_2), V), (\sigma, out_2))\}$

$\textit{ideal-view}_{2,AND}((b_1, b_2), (a_1, a_2), out_2) = do \{$
$\quad V \leftarrow S_{2,OT_4^1}((b_0, b_1), out_2);$
$\quad let\ s_1 = out_2 \oplus (a_1 \oplus b_1) \wedge (a_2 \oplus b_2);$
$\quad return(((b_1, b_2), V), (s_1, out_2))\}$

E Malicious OT_2^1 DDH adversaries

The adversaries we construct to play against the DDH assumption in Section 6.3.2 are given below. For the case where $\sigma = 0$ if (h, s, t) is a DDH tuple then (h_0, h_1, a, b_0, b_1) the adversary constructs is the same as in the real view when $\sigma = 0$. On the other hand if (h, s, t) is such that $s = g^r$ and $t = g \cdot h^r$ for some r then (h_0, h_1, a, b_0, b_1) constructed by the adversary is the same as if it were constructed by the simulator (when $\sigma = 0$). Moreover if the adversary receives a DDH tuple then the output $x_0 = \frac{z_0}{w_0^{\alpha_0}}$ is the same as the real execution of the protocol whereas if the adversary receives a non-DDH tuple then output of the adversary is the same as in the ideal model. The intuition is analogous for the case $\sigma = 1$. For completeness we provide the adversaries for both cases here.

$DDH\text{-}\mathcal{A}_{\sigma=0}(M, z, (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), D, h, a, t) = do\ \{$
$\alpha_0 \leftarrow uniform(|G|);$
$let\ h_0 = g^{\alpha_0};$
$let\ h_1 = h;$
$let\ b_0 = a^{\alpha_0};$
$let\ b_1 = t;$
$((in_1, in_2, in_3), s) \leftarrow \mathcal{A}_1(M, h_0, h_1, a, b_0, b_1, z);$
$\_ \leftarrow assert(in_1 = h_0 \otimes h_1^{-1} \wedge in_2 = a \wedge in_3 = b_0 \otimes b_1^{-1});$
$(((w_0, z_0), (w_1, z_1)), s') \leftarrow \mathcal{A}_2(h_0, h_1, a, b_0, b_1, M, s);$
$let\ x_0 = z_0 \otimes w_0^{-\alpha_0}$
$adv_{out} \leftarrow \mathcal{A}_3(s');$
$D(adv_{out}, x_0)\}$

$DDH\text{-}\mathcal{A}_{\sigma=1}(M, z, (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), D, h, a, t) = do\ \{$
$\alpha_1 \leftarrow uniform(|G|);$
$let\ h_1 = g^{\alpha_1};$
$let\ h_0 = h;$
$let\ b_1 = a^{\alpha_1} \otimes g;$
$let\ b_0 = t;$
$((in_1, in_2, in_3), s) \leftarrow \mathcal{A}_1(M, h_0, h_1, a, b_0, b_1, z);$
$\_ \leftarrow assert(in_1 = h_0 \otimes h_1^{-1} \wedge in_2 = a \wedge in_3 = b_0 \otimes b_1^{-1});$
$(((w_0, z_0), (w_1, z_1)), s') \leftarrow \mathcal{A}_2(h_0, h_1, a, b_0, b_1, M, s);$
$let\ x_1 = z_1 \otimes w_1^{-\alpha_1}$
$adv_{out} \leftarrow \mathcal{A}_3(s');$
$D(adv_{out}, x_1)\}$

F Guide to the source theory files

For brevity we only presented some of the formalisation in the main text. However all definitions and statements of security have been formalised in Isabelle/HOL. Here we provide a guide to the reader to navigate the source files.

- **Semi_Honest_Defs.thy** formalises two party security in the semi-honest model (Section 3).
- **OT_Functionalities.thy** formalises the definitions of the variants of OT we use.
- **ETP.thy** formalises ETPs at an abstract level (Section 4.1).
- **ETP_OT.thy** formalises the general construction of the OT_2^1 constructed from ETPs (Section 4.2).
- **ETP_OT_RSA.thy** formalises the RSA instantiation of the the proof from **ETP_OT.thy** (Section 4.4).
- **OT14.thy** formalises the construction of OT_4^1 (Appendix C).
- **GMW.thy** formalises the two party GMW protocol (Section 5).
- **Malicious_Defs.thy** formalises malicious security in the two party setting (Section 6.1).
- **Malicious_OT.thy** formalises the OT_2^1 construction that provides malicious security (Section 6.3).
- **Uniform_Sampling.thy** formalises numerous one time pad constructions used in our proofs.
- **Cyclic_Group_Ext.thy** extends the formalisation of cyclic groups from CryptHOL, providing results we require in this work.
- **DH_Ext.thy** extends the formalisation from [5, 26] of the Diffie Helman assumption to include the variants we require in this work.
- **Number_Theory_Aux.thy** formalises various results from number theory we require.