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

***Abstract*—Large scale implementations of Multi-Party Computation (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. CryptHOL provides a formal probabilistic programming framework, embedded in the Isabelle/HOL proof assistant, for reasoning about cryptographic proofs.**

Oblivious Transfer is central to realising MPC. In this work, we formalise protocols for 1-out-of-2 Oblivious Transfer (OT_2^1) 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 GMW protocol.

Our semi-honest OT_2^1 is constructed from Extended Trapdoor Permutations (ETP), we first prove the general construction secure and then instantiate for the RSA collection.

***Index Terms*—Multi-Party Computation, Oblivious Transfer, Formal Verification, Isabelle/HOL, Malicious Security**

I. Introduction

Multi-Party Computation (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 [22] where he posed and proposed the first solution to the problem and shortly after Goldreich et al. [11] where the GMW protocol was introduced. 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 [5] and private statistical computation, e.g., using Sharemind [4].

The potential large scale implementation of MPC means it is of high importance to have guarantees of correctness beyond the paper proofs which cryptographers have relied on for many years. Despite the obvious value of paper proofs, it is important to examine them under the lens of formal verification in particular when modularisation and composition is involved. It would be highly desirable to have 'end-to-end' machine checked verification of the entire process. We believe the work required to achieve this can be partitioned into two categories: 1) formally verifying the security properties of the underlying cryptography, 2) formally verifying the implementation of the protocols and algorithms. We consider the first of these.

Oblivious Transfer (OT), a two party protocol, is at the heart of many MPC protocols. We consider 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. Our study of it here is motivated by its almost universal use across MPC and its central role to Garbled Circuits [23] and GMW. 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 to GMW.

Security of MPC protocols is often proven in the simulation-based paradigm, where one simulates the real world adversaries in an ideal world where security is guaranteed by construction.

There are two definitions of security of differing strength. 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 [14] 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 [10] and Lindell and Hazay [13].

We formalise our proofs in the theorem prover Isabelle/HOL using CryptHOL [3] which provides an embedding of a probabilistic programming framework. We model the views (real and ideal) of parties in protocols as probabilistic programs and define semi-honest and malicious security with respect to these programs. We then instantiate the views for the protocols we consider and prove our definitions of security are satisfied. An increasing number of proofs have been completed using CryptHOL [6, 7, 18, 19].

A. Contributions

We split our contributions based on the security model they relate to. The diagram in Fig. 1 shows how our formal theories and contributions relate to each other.

a) Semi-honest model:

- We provide formal abstract simulation-based definitions for the security of MPC. (Section III-A)
- We prove security of a OT_2^1 protocol [8] constructed from a general Extended Trapdoor Permutation (ETP). We instantiate this for the RSA collection. (Section IV)
- We show how OT_4^1 is constructed from OT_2^1. (See the formalisation)

- We prove security of AND and XOR gates in the GMW protocol. (Section V)

b) Malicious model:

- We provide formal abstract definitions of security. (Section VI)
- We prove security with respect to these definitions of a protocol that realises OT_2^1 [13]. (Section VI)

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 [7] 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. Our complete formalisation can be found at [2].

B. Related work

Semi-honest security has been considered in EasyCrypt in [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 III-A we prove their definitions are equivalent to ours. The challenge faced in the 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 [12] where the authors prove security of Maurer's protocol [20]. 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 as in proving the meta theorem a new approach is proposed; nonetheless one would prefer a completely formalised approach. Moreover, only information theoretic security was considered whereas we consider reduction based proofs. Also [12] does not consider the asymptotic case unlike our work.

Work using the CryptHOL framework to date is limited meaning this work provides a considerable contribution for others to learn from. Originally CryptHOL was used for game-based proofs [19] and has recently been used for constructive cryptography [18] and commitment schemes and Σ-protocols [7]. Butler et al. [6] used CryptHOL for MPC protocols in the semi-honest model including a proof of the Noar-Pinkas OT_2^1 protocol, 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.

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

(i) We extend the work of Butler et al. [6] by giving more modular proofs of oblivious transfer protocols in the semi-honest setting, building OT_4^1 from OT_2^1, and proving security of the GMW protocol.

(ii) Moreover we investigate the malicious security model for MPC by proving the OT_2^1 of [13]. This complements the work of Haagh et al. [12] in that the formalisation consider 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.

C. Outline of Paper and Formalisation

An outline of the paper can be seen in Fig. 1. Here dashed boxes represent abstract definitional theories — these are mainly definitional however we also prove some general lemmas that reduce the workload in the instantiated proofs. Solid boxes represent proofs of security and arrows represent imported theories. We provide formalisation for the whole of Fig. 1.

The major advantage of providing and then instantiating abstract definitional theories is a human checker only needs to verify that these definitions correspond to the correct security notions.

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 IV-E.

II. CryptHOL and Isabelle Background

In this section we follow [7] and briefly introduce the Isabelle notion we use throughout and then highlight and discuss some important aspects of CryptHOL. For more detail on CryptHOL see [3]. The full formalisation is available at [16].

A. Isabelle notation

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 $a \Rightarrow b$ is the type of functions that takes an input of type a and outputs an element of type b. 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 the group.

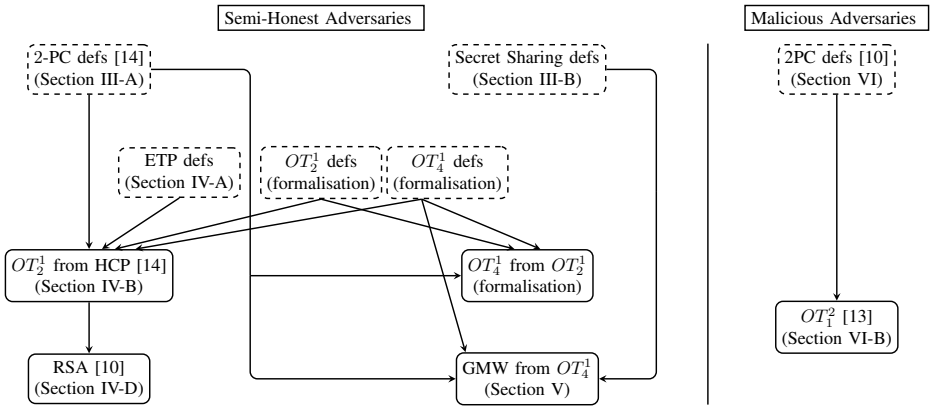

Fig. 1. Outline of the formalisation for the paper.

B. CryptHOL

CryptHOL [3] is a framework for reasoning about cryptography in the computational model that is embedded inside the Isabelle/HOL theorem prover. It allows the prover to write probabilistic programs and reason about them. The computational model is based on probability theory and in particular uses probabilistic programs to define security — this can be seen for the construction of games in the game-based setting or the real and ideal views in the simulation-based setting.

To build the probabilistic programming framework CryptHOL uses the existing probability theory formalised inside Isabelle to define discrete probability distributions called sub probability mass functions (of type $spmf$). These can be thought of as probability mass functions with the property they do not have to sum to one — we can lose some probability mass. This allows us to model failure events and assertions.

1) Writing probabilistic programs: CryptHOL provides some, easy-to-read, Haskell-style do notation to write probabilistic programs where $\mathbf{do}\{x \leftarrow p;\ f(x)\}$ is the probabilistic program that samples from the distribution p and returns the $spmf$ produced by f. The do notation desugars to $p \rhd (\lambda x.\ f(x))$. We can also return an $spmf$ using the monad operation $return$. See Fig. 5 for an example.

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) Sampling: Sampling from sets is important in cryptography. CryptHOL gives an operation $uniform$ 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 $\{..<q\}$ and by $coin$ we denote the uniform sampling from the set $\{True, False\}$ — a coin flip.

3) Probabilities: We must also be able to reason about the probability of events occurring. So, $\mathcal{P}[Q = x]$ denotes the subprobability mass the spmf Q assigns to the event x.

4) 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 IV-E.

5) 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 constants and then instantiate these definitions for explicit protocols and prove the security properties as theorems.

III. 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. We follow the definitions of security given by Lindell [15].

A. Two party protocol security

A functionality is a function that maps inputs to desired outputs for a defined protocol problem. In this section we show our formalisation for the case where the functionality is deterministic (as 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.

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, that is the output of the real view and the simulator are indistinguishable. This simulation of the real running of the protocol means no information is leaked during its execution.

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*

Fig. 2. The locale constants for defining semi-honest security for deterministic functionalities.

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

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

where $w \in \{x, y\}$ and is dependent on which view we are considering, 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.

A protocol π is said to securely compute f in the presence of a semi-honest adversary if there exist probabilistic polynomial time algorithms (simulators) S_1, S_2 such that,

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

$$\{S_2(1^n, y, f_2(x, y))\} \stackrel{c}{\equiv} \{view_2^{\pi}(x, y, n)\} \quad (2)$$

where $\stackrel{c}{\equiv}$ denotes computational indistinguishability. One limitation of CryptHOL is the inability to reason about polynomial run time, or in fact the *feasibility* of adversaries at all, thus one must manually verify that the simulators and any constructed adversaries run in the required time. In [6] the authors defined computational indistinguishability in Isabelle modulo run-time constraints, however here we follow the approach of [17] and consider advantages. Advantages can be thought of as analogous to indistinguishability as they define the probability (the advantage) of a distinguisher distinguishing two probability distributions — in our case the real and simulated views.

We begin by fixing the required constants to make our definitions, we do this using Isabelle's module system (called locales). Fig. 2 shows the locale *semi-honest-det* (for semi-honest deterministic functionalities); it fixes *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.

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. The constant *protocol* is a probabilistic program that describes the execution of the protocol, returning each parties' output for the protocol. The exact probabilistic program defining the protocol will depend on which protocol we are using to realise a functionality.

In the case of OT_2^1 the functionality is defined as.

$$funct_{OT_2^1}((m_0, m_1), \sigma) = return(\_, if\ \sigma\ then\ m_1\ else\ m_0). \quad (3)$$

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

To formally prove security we are required to construct a simulator for each party that receives as input the input message of the party and the output of the party, as given by the functionality. To construct the ideal view we must sample from the functionality and use the binding operator ($\triangleright$) to hand it to the simulator as follows:

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

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

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

Information theoretic security requires the real and simulated views to be equal. We define this for Party 1 below:

Definition 2 (Information theoretic security, for Party 1):

$$inf\_theoretic\_P_1(m_1, m_2) \equiv (R_1(m_1, m_2) = ideal_1(m_1, m_2))$$

We make the analogous definition for Party 2.

When information theoretic security cannot be proven we show that the probability of distinguishing the real and simulated views is small. In the asymptotic setting we show this probability is a negligible function in the security parameter. We make the initial definitions in terms of the probability (advantage) a distinguisher has of distinguishing the real and simulated views. We define the advantage of a distinguisher, D, for Party 1 as follows.

Definition 3:

$$adv\_P_1(m_1, m_2, D) \equiv (|\mathcal{P}[(R_1(m_1, m_2) \triangleright D) = True] - \mathcal{P}[(ideal_1(m_1, m_2) \triangleright D) = True]|)$$

The right hand side of the above definition is the formalised version of writing $|Pr[D(R_1(m_1, m_2)) = 1] - Pr[D(S_1(m_1, f_1(m_1, m_2))) = 1]|$, which is more commonly seen in the literature. We also provide the analogous definition for Party 2.

The definitions in this section have been extracted from [6] and formalised in a modular way so they can be imported into any theory.

In [1] Almeida et al. 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.

B. Secret sharing schemes

Secret sharing schemes [21] are at the core of MPC protocols. To formalise such schemes we provide two constants *share* and *reconstruct* that define the sharing scheme and a third, a set, *evaluate* which represents the set of functions we wish to realise (in our instantiation of the GMW these are AND and XOR). We give their types below.

$$share \ :: \ `a \Rightarrow (`share \times `share) \ spmf \tag{4}$$

$$reconstruct \ :: \ (`share \times `share) \Rightarrow `a \ spmf \tag{5}$$

$$evaluate \ :: \ (`a \Rightarrow `a \Rightarrow `a \ spmf) \ set \tag{6}$$

The correctness property requires that reconstructing a shared input returns the original input.

Definition 4 (Correctness on secret sharing):

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

In the instantiations we define the set *evaluate* and prove correctness for all its elements with respect to Definition 1. The security properties of the protocols that use secret sharing are proven with respect to the definitions in the previous section.

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

IV. 1-out-of-2 OT using Enhanced Trapdoor Permutations

In this section we present our formalisation of the protocol realising OT_2^1 using a general Enhanced Trapdoor Permutation (ETP) [8]. In the proof of security one must assume that such ETPs and thus the associated Hard Core Predicates (HCPs) exist.

A. ETPs and HCPs

We recap the paper based definitions of an ETP and refer the reader to [15] (Section 4.3) and [10] (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 with which the inverse can be obtained easily.

- $I(n)$ samples an index α of a permutation, f_α, as well as a corresponding trapdoor τ for the permutation, $(\alpha, \tau) \leftarrow I(n)$.
- $S(\alpha)$ 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 [15] and [10] 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.

Associated with an ETP is an HCP, B. We assume such a B exists and fix it in the locale,

$$B \ :: \ bitstring \Rightarrow `range \Rightarrow bool.$$

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 constants: I, $domain$, $range$, f and B and defines S as uniformly sampling from the range. We make the following assumptions on the constants.

Assumption 1: Assumptions made on the fixed constants I, $domain$, $range$, f and B to form an ETP.

1) $domain(\alpha) = range(\alpha)$
2) $finite(range(\alpha))$
3) $range(\alpha) \neq \{\}$
4) $bij\_betw(f, domain(\alpha), range(\alpha))$

Here α is always sampled from I and $bij\_betw(f, A, B)$ denotes that f provides a bijection between the sets A and B — as we have $domain(\alpha) = range(\alpha)$ this implies f is a permutation.

To formally define the security property of HCPs we require 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 first define the HCP game as follows,

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

We then 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 $\mathcal{A}$ receives α as input and σ and b_σ on its advice tape. In addition we must pass x to $\mathcal{A}$ also, this is because we do not carry around the randomness given to S however we must allow the adversary access to x.

B. Realising OT_2^1 using ETPs

We consider the OT_2^1 protocol from [8] which is described in Protocol 1.

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

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

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

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

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

Using this definition and the functionality given in Eq. 3 we show correctness of Protocol 1.

Theorem 1:

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

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.

C. 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 will mainly 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 proof method from [14] (Section 4.3).

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

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

Fig. 3. The real and simulated views for Party 2.

To show security for Party 2 we must show that $adv\_P_{2,OT_2^1,ETP}$ is negligible. In the first instance we show the advantage is less than or equal to $2.adv_{HCP}$. When we instantiate the security parameter we show that $adv\_P_{2,OT_2^1,ETP}$ is negligible (as the HCP advantage is negligible), we discuss this in more detail in Section IV-D. The simulator, along with the real view, for Party 2 is given in Fig. 3.

For the case where $b_{1-\sigma} = False$ (formally this is $(if\ \sigma\ then\ b_0\ else\ b_1) = False$) we have information theoretic security.

Lemma 1: Assume $b_{1-\sigma} = False$ 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 $adv\_P_{2,OT_2^1,ETP}((b_0, b_1), \sigma, D) = 0$ when $b_{1-\sigma} = False$. It is left to consider the case where $b_{1-\sigma} = True$. We construct an adversary, $\mathcal{A}_{HCP}$ (shown in Fig. 4), that breaks the HCP assumption if D can distinguish the real and simulated views — that is we show a reduction to the HCP assumption. We show that the advantage this adversary has against the HCP assumption is the same as the advantage a distinguisher has in distinguishing the real and simulated views. The analogous paper proof would look for a contradiction by constructing the HCP adversary after assuming a distinguisher can distinguish the views. We bound the advantage of Party 2 as follows.

Lemma 2: Assume $b_{1-\sigma} = True$ then we have

$$
\begin{aligned}
&adv\_P_{2,OT_2^1,ETP}((b_0, b_1), \sigma, D) =\\
&\qquad\qquad 2 \cdot adv_{HCP}(\mathcal{A}_{HCP}, \sigma, b_\sigma, D)
\end{aligned}
$$

Where '$\cdot$' denotes multiplication of the real numbers.

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

Fig. 4. The adversary used to break the HCP game when showing security for Party 2.

The proof of Lemma 2 is technical and involved. We formally defined 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 [15]. One proof step was formally more difficult to reason about than the others. This is the first step of the proof in [15] (first equality of p14) where we are required to split the probability of $\mathcal{A}_{HCP}$ winning the HCP game into two cases, dependent on the coin flip $\mathcal{A}_{HCP}$ makes ($\beta_{1-\sigma}$). 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 (in particular results on integration) theory formalised in Isabelle. More precisely, we are required to prove that extracting the sample from the probabilistic program is legitimate so the cases can be reasoned about.

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

Theorem 2:

$$adv\_P_{2,OT_2^1,ETP}((b_0, b_1), \sigma, D) \leq 2.adv_{HCP}(\mathcal{A}_{HCP}, \sigma, b_\sigma, D) \quad (7)$$

For Party 1 we are able to construct a simulator, $S_{1,OT_2^1,ETP}$, in the same manner as in [15] and show it is equal to the real view.

Theorem 3: For Party 1 we have information theoretic security.

$$inf\_theoretic\_P_{1,OT_2^1,ETP}((b_0, b_1), \sigma)$$

Together Theorems 2 and 3 show Protocol 1 is secure in the concrete setting.

D. Instantiating for RSA

It is known that the RSA collection of functions provides an ETP (see [9] Section 2.4.4.2 together with [10] Section C.1). We formalise this RSA collection and instantiate it for Protocol 1. We fix as a constant 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. The permutation here is,

$$f((N, e), x) = x^e\ mod\ N \quad (8)$$

for appropriately chosen N and e.

To show security 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 Assumption 1 holds in the new context. The most challenging of these assumptions to prove is that the RSA function (Equation 8) 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.

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 α is the index outputted by I, then we have

$$bij\_betw(f(\alpha), domain(\alpha), range(\alpha)).$$

This is the main proof statement we require to import our proof from the general case to the RSA instantiation. One assumption we must carry over, of course, is the fact that such a HCP (B) exists for the RSA collection. The security results are as follows.

Theorem 4: For the ETP constructed from the RSA collection we have

$$inf\_theoretic\_P_{1,OT_2^1,RSA}((b_0, b_1), \sigma)$$

and

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

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 in full from scratch. Subsequent leveraging of the module system 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 Assumption 1 are valid. In fact no security results need to be proven at all in future instantiations. This highlights a main advantage of working with proof assistants such as Isabelle, future workload can be significantly reduced using a modular approach.

E. The RSA instantiation in the asymptotic setting

In this section we outline how we prove security in the asymptotic setting.

Reasoning over the security parameter in the asymptotic setting allows a closer equivalence to the pen and paper security properties. One area where this is realised is in the ability to more accurately define hardness assumptions. For example in Theorem 4 we could only bound the advantage of Party 2 by the HCP advantage. While this implies security we would like to explicitly make the assumption that the HCP

advantage is negligible and thus show the advantage for Party 2 is also negligible.

We provide proofs in the asymptotic setting for all protocols we consider — here we present the instantiation of RSA by way of example.

In this instance we introduce the security parameter as an input to the set of primes we have fixed, specifically we change the type of $prime\_set$ from $nat\ set$ to $nat \Rightarrow nat\ set$. Thus the set of primes is now parametrised by the security parameter — a natural[1]. Intuitively the security parameter can be thought of as selecting which set of primes the ETP is defined over.

After importing the concrete setting parametrically for all n, we see that all algorithms now depend explicitly on the security parameter. Moreover, due to Isabelle's module structure we are able to use results proven in the concrete setting in our newly constructed asymptotic setting. 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 5: The RSA instantiation of Protocol 1 is correct.

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

The security parameter only appears as inputs to functions where it is used. Equation 3 shows that the security parameter is never required to define $funct_{OT_2^1}$. Security is shown by the following Theorem.

Theorem 6: For Party 1 we have information theoretic security, that is

$$inf\_theoretic\_P_{1,OT_2^1,RSA}(n,(b_0,b_1),\sigma)$$

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

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

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

V. Formalising the GMW protocol

The GMW protocol provides a method to securely compute any boolean circuit. It does so by providing a method for computing gates in the circuit securely — AND and XOR gates are sufficient. 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 — AND and XOR form a universal set from which we can realise the whole of MPC.

[1]The algorithm I samples from this set, thus it is dependent on the security parameter.

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

Fig. 5. The sharing and reconstruction algorithms used in the GMW protocol

A. 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. The formal algorithms for this can be seen in Fig. 5.

To show correctness we show that reconstructing a shared input results in the original input.

Theorem 7:

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

B. Securely computing AND and XOR gates

We show how to securely compute an XOR and AND gate using the GMW protocol. Assume Party 1 has input x and Party 2 has input y, after sharing and sending the other party the appropriate share Party 1 holds the shares (a_1, a_2), and Party 2 holds the shares (b_1, b_2) — that is $x = a_1 \oplus b_1$ and $y = a_2 \oplus b_2$.

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

To achieve secure computation of an AND gate we require OT_4^1. We take the protocol that realises OT_k^1 from [10] (Section 7.3.3, p640) and adapt it for the case of OT_4^1. The functionality for OT_4^1 is defined as,

$$\begin{aligned} &funct_{OT_4^1}((b_{0,0},b_{0,1},b_{1,0},b_{1,1}),(c_0,c_1)) \\ &\qquad = return((),b_{c_0,c_1}). \end{aligned} \quad (9)$$

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 modularity we make assumptions on the security of OT_2^1 in the locale. We assume the security results of the Noar Pinkas OT_2^1 which is used in practical implementations of GMW, that is we assume: correctness, information theoretic security for Party 2 and bound the advantage of Party 1. Using these assumptions we prove the following security theorems for the construction of OT_4^1 from [10].

Theorem 8: Assume that

$$adv\_P_{1,OT_2^1}((m_0,m_1),\sigma,D) \leq P_1\, adv_{OT_2^1}$$

then we have

$$adv\_P_{1,OT_4^1}(M,C,D) \leq 3 \cdot P_1 adv_{OT_2^1}.$$

Theorem 9: Assume that

$$inf\_theoretic\_P_{2,OT_2^1}((m_0, m_1), \sigma)$$

then we have

$$inf\_theoretic\_P_{2,OT_4^1}((M, C).$$

Where M and C represent the inputs of OT_4^1.

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 $b_1 \oplus b_2$.

There is no need for any communication between the parties thus security is achieved. Correctness comes from the commutativity of the XOR operation.

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

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

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 xored together give the desired result. Intuitively, security comes from Party 1 constructing all possible results of the computation (in Step 1) and masking it with the random sample σ. The parties then use OT_4^1 to transfer over one and only one of the possible values, the value that Party 2 can *decrypt* to form their share.

We define probabilistic programs for these protocols and their respective functionalities; $protocol_{XOR}$, $funct_{XOR}$, $protocol_{AND}$, $funct_{AND}$ — these take the shares held by the respective parties as inputs.

We define $evaluate = \{evaluate_{XOR}, evaluate_{AND}\}$ as the set of functions we wish to consider — these are defined over the original inputs (e.g. $evaluate_{AND}(x, y) = return(x \wedge y)$). We prove Protocol 2 and 3 are correct with respect to the secret sharing. For example for the AND gate construction we prove Lemma 4, that is sharing the inputs, executing the protocol and reconstructing gives the desired evaluated result.

Lemma 4:

$$\begin{aligned} &(share_{GMW}(x) \triangleright (\lambda(s_1, s_2).\ share_{GMW}(y) \triangleright \\ &\quad (\lambda(s_3, s_4).\ protocol_{AND}((s_1, s_3), (s_2, s_4)) \triangleright \\ &\quad\quad (\lambda(S_1, S_2).\ reconstruct_{GMW}(S_1, S_2)))) \\ &\quad\quad\quad\quad = evaluate_{AND}(x, y)). \end{aligned}$$

We show security for both gates. There is no communication between the parties when computing the XOR gate thus security comes easily. Here we focus on the AND gates security.

We have information theoretic security of OT_4^1 for Party 2 thus we show information theoretic security for Party 2 of the AND gate. That is we construct $S_{2,AND}$ such that Theorem 10 holds.

Theorem 10: Assuming the security results on OT_1^4 from Section **??** we have,

$$inf\_theoretic\_P_{2,AND}((a_1, a_2), (b_1, b_2)).$$

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

Theorem 11: Assume that

$$adv\_P_{1,OT_4^1}(M, (c_0, c_1), D) \leq P_1 adv_{OT_4^1}$$

then we have

$$adv\_P_{1,AND}((a_1, a_2), (b_1, b_2), D) \leq P_1 adv_{OT_4^1}.$$

Our proof of Theorem 11 is similar in construction to the reduction we show for Party 1 of the OT_2^1 protocol in Theorem 8.

Theorems 10 and 11 show security in the semi-honest model for the AND gate construction.

VI. Formalising Malicious Security

We now consider the malicious setting. We first formally define malicious security and then prove an OT_2^1 protocol secure with respect our definitions.

A. Formalising the definitions

In the malicious security model an adversary fully corrupts one of the parties (recall we work in the two party setting) 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 equal to its proper 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 [10] (Section 7.2.3 p 626).

A protocol is said to be secure if the adversary's behaviour is limited to the three actions given above. We consider the malicious security definitions from [10] (Section 7.2.3) and [13] (Section 2.3.1 p 24/25) where an ideal and real world are considered.

The ideal model uses a trusted party that ensures security by definition — we let x be the input of Party 1, y be the input of Party 2 and z be the auxiliary input[2] available to the adversary. The ideal model is constructed as follows ([13]):

- **Send inputs to trusted party:** The honest party sends its received input to the trusted party. The input for the corrupted party is outputted by the adversary and given to the trusted party (it could be abort, the adversary chooses the input based on the original input and z).
- **Early abort:** If the trusted party receives abort from the corrupted party it sends abort to both parties and the execution is terminated.
- **Trusted party computation:** The trusted party computes the functionality using the inputs provided by both parties and sends the corrupted party its output.
- **Adversary aborts or continues:** The adversary, upon receiving its output, instructs the trusted party to abort or continue. If abort is sent the execution is terminated, if continue is sent the trusted party sends the honest party its output.
- **Outputs:** The honest party outputs the output it received from the trusted party, the corrupted party outputs nothing. The adversary outputs any arbitrary function of the initial input, auxiliary input, and the output given to it by the trusted party.

The output of the ideal model, when Party i is corrupted, is denoted as $IDEAL_{f,\mathcal{A}(z),i}(x,y)$. This is 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 world such that the $IDEAL_{f,S(z),i}(x,y)$ and $REAL_{\pi,\mathcal{A}(z),i}(x,y)$ are indistinguishable.

Here we show our formalisation of the security definitions for Party 2, the formalisation for Party 1 is analogous. The definition of correctness is the same as in the semi-honest model, we do not redefine it here.

To make the definitions we fix as constants in the locale; the functionality $funct$, the real execution mal_{real_2}, and the simulator $S = (S_1, S_2)$ — this is the simulator that interacts in the ideal model. In a security proof we must explicitly define these constants.

For clarity we define the trusted party as the functionality.

$$trusted\_party(x,y) = funct(x,y) \tag{10}$$

[2]This is a priori information.

$$\begin{aligned}
&ideal_model_2(x,y,z,\mathcal{A}) = do\ \{\\
&\quad let\ (\mathcal{A}_1,\mathcal{A}_2) = \mathcal{A};\\
&\quad (y', aux_{out}) \leftarrow \mathcal{A}_1(y,z);\\
&\quad (out_1, out_2) \leftarrow trusted_party(x,y');\\
&\quad out_2' \leftarrow \mathcal{A}_2(y', z, out_2, aux_{out});\\
&\quad return(out_1, out_2')\}
\end{aligned}$$

Fig. 6. The formal definition of the ideal model for a corrupt Party 2

Our formalisation of the ideal model for Party 2 is defined by the probabilistic program given in Fig. 6. 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, this is because they are covered by the type of $spmf$. When the adversary provides output in the probabilistic program it could also output None, this captures the abort process as it terminates the probabilistic program with no output.
2) The first part of the adversary $\mathcal{A}_1$ outputs the input it wishes to give to the trusted party (as in the description of the ideal model) as well as some auxiliary output (aux_{out}). This is not described explicitly in the ideal model in the literature however is required for the adversary to maintain state; we split one adversary into two parts, thus we must allow the two parts to share state.

Definition 6: We define the ideal view of Party 2 as follows,

$$\begin{aligned}
&mal_{ideal_2}(x,y,z,(S_1,S_2),\mathcal{A}) \equiv\\
&\qquad ideal_model_2(x,y,z,(S_1(\mathcal{A}),S_2(\mathcal{A})))
\end{aligned} \tag{11}$$

Here $\mathcal{A}$ consists of a tuple of algorithms, one for each round of the protocol.

As in the semi-honest case we either show information theoretic security or show the views are indistinguishable — in which case we refer to the advantage that a distinguisher has of distinguishing them. For information theoretic security we require equality between the views.

Definition 7 (Information theoretic security for Party 2):

$$\begin{aligned}
&mal_{inf_th_2}(x,y,z,S,\mathcal{A}) \equiv\\
&\qquad (mal_{ideal_2}(x,y,z,S,\mathcal{A}) = mal_{real_2}(x,y,z,\mathcal{A}))
\end{aligned}$$

Here mal_{real_2} describes the real view of Party 2. This is the probabilistic program of the execution of the protocol where all messages sent by Party 2 are sent by the adversary.

The advantage of a distinguisher to be is defined as follows.

Definition 8 (The advantage of a distinguisher for Party 2):

$$\begin{aligned}
&mal_{adv_2}(x,y,z,S,\mathcal{A},D) =\\
&|\mathcal{P}[(mal_{real_2}(x,y,z,\mathcal{A}) \rhd\ (\lambda view.\ D(view)) = True] -\\
&\mathcal{P}[(mal_{ideal_2}(x,y,z,\mathcal{A}) \rhd (\lambda view.\ D(view))) = True]|
\end{aligned}$$

The work of Haagh et al. [12] formalises the same malicious model (active security model) in EasyCrypt, however as discussed in Section I-B a meta (paper) theorem was required to link the formalisation to the paper definitions.

B. A protocol realising OT_2^1 in the malicious setting

In this section we show the definitions we provide for malicious security are satisfied by the OT_2^1 protocol from [13] (p190). This protocol is considered in the hybrid model as it uses a call to a Zero Knowledge Proof of Knowledge (ZKPOK) functionality for the DH relation (F_{ZK}^{DH}). Specifically we have,

$$F_{ZK}^{DH}((h,a,b),(((h',a',b'),r)) = \\ return(a = g^r \wedge b = h^r \wedge (h,a,b) = (h',a',b'),\_). \quad (12)$$

Here the output for Party 1 is a boolean dependent on whether the relation is true, and Party 2 gets no output.

The protocol uses a cyclic group G (where it is assumed the DDH assumption holds) of order q with generator g and is run as follows:

Protocol 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 $\alpha_0, \alpha_1, r \leftarrow \{1,...,q\}$ and computes $h_0 = g^{\alpha_0}$, $h_1 = g^{\alpha_1}$, $a = g^r$, $b_0 = h_0^r \cdot g^\sigma$ and $b_1 = h_1^r \cdot g^\sigma$.
3) S checks $(h_0, h_1, a, b_0, b_1) \in G^5$, otherwise it aborts.
4) Let $h = h_0/h_1$ and $b = b_0/b_1$. R proves to S that (h,a,b) is a DH tuple, using ZKPOK. That is R proves the relation
$$\mathcal{R}_{DH} = \{((h,a,b),r).\ a = g^r \wedge b = h^r\}$$
5) If S accepts the proof it continues and uniformly samples $u_0, v_0, u_1, v_1 \leftarrow \{1,...,q\}$, and computes (e_0, e_1) and sends the tuple to R:
   $e_0 = (w_0, z_0)$ where $w_0 = a^{u_0} \cdot g^{v_0}$ and $z_0 = b_0^{u_0} \cdot h_0^{v_0} \cdot m_0$.
   $e_1 = (w_1, z_1)$ where $w_1 = a^{u_1} \cdot g^{v_1}$ and $z_1 = (\frac{b_1}{g})^{u_1} \cdot h_1^{v_1} \cdot m_1$.
6) R outputs $\frac{z_\sigma}{w_\sigma^{a_\sigma}}$ and S outputs nothing.

Here '$\cdot$' denotes multiplication in the group and $\frac{a}{b}$ denotes multiplication by the inverse of b in the group (in the case where $a, b \in G$).

Correctness of the protocol can be seen if one splits the proof into the cases on the value of σ. Intuitively security for the receiver is upheld because σ is sent only as an exponent of the generator which is masked by random group element and the receiver can learn at most one of the sender's messages due to the construction of the DDH tuple, which the sender is satisfied with after the ZKPoK.

C. Proving OT_2^1 secure in the malicious setting

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

Theorem 12: 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 must first formalise a variant of the DDH assumption and prove it is at least as hard as the traditional DDH assumption. The security of the Sender is reduced to this assumption.

1) 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 (where x, y and z are random samples), the variant states that (h, g^r, h^r) and $(h, g^r, h^r \cdot g)$ are hard to distinguish (where $h \in G$, and g is the generator of G). We formalise this variant of the assumption and prove it is as hard as the original DDH assumption in Lemma 5.

Lemma 5:

$$DDH\_adv_{var}(\mathcal{A}) \leq \\ DDH\_adv_{orig}(\mathcal{A}) + DDH\_adv_{orig}(\mathcal{A}'(\mathcal{A})) \quad (13)$$

Where DDH_adv_{orig} is the original DDH advantage (formalised in [19]), DDH_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. Lemma 5 shows the variant is at least as hard as the original assumption.

We now show security of each party in turn.

2) Party 1: To show security for Party 1 we make a case split on σ. We construct $S_1 = (S_{1,P_1}, S_{2,P_1})$ as given in [13] and construct an adversary for each case ($DDH\_\mathcal{A}_{\sigma=1}$, $DDH\_\mathcal{A}_{\sigma=0}$) that *breaks* the variant of the DDH assumption and show:

Lemma 6: Assume $\sigma = 0$ then we have,

$$mal_{adv_1}((m_0,m_1),\sigma,z,S_1,\mathcal{A},D)) = \\ DDH\_adv_{var}(DDH\_\mathcal{A}_{\sigma=0}(\mathcal{A},D)).$$

Lemma 7: Assume $\sigma = 1$ then we have,

$$mal_{adv_1}((m_0,m_1),\sigma,z,S_1,\mathcal{A},D)) = \\ DDH\_adv_{var}(DDH\_\mathcal{A}_{\sigma=1}(\mathcal{A},D)).$$

We note here we use 1 as an encoding for $True$ and 0 as an encoding for $False$. Using Lemma 5 we bound the malicious advantages by the traditional DDH assumption advantage.

Lemma 8: Assume $\sigma = 0$ then we have,

$$mal_{adv_1}((m_0,m_1),\sigma,z,S_1,\mathcal{A},D)) \leq \\ DDH\_adv_{orig}(DDH\_\mathcal{A}_{\sigma=0}(\mathcal{A},D)) + \\ DDH\_adv_{orig}(\mathcal{A}'(DDH\_\mathcal{A}_{\sigma=0}(\mathcal{A},D)))$$

Lemma 9: Assume $\sigma = 1$ then we have,

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

Due to the case split on σ the security result in the concrete setting is quite convoluted and tricky to read, hence, here we

$S_{1,P_2}((\mathcal{A}_1,\mathcal{A}_2,\mathcal{A}_3),\sigma,z) = do\ \{$
$(h_0,h_1,a,b_0,b_1) \leftarrow \mathcal{A}_1(\sigma);$
$\_ \leftarrow assert_{spmf}(h_0,h_1,a,b_0,b_1 \in G);$
$((in_1,in_2,in_3),r) \leftarrow \mathcal{A}_2(h_0,h_1,a,b_0,b_1);$
$let\ (h,a,b) = (\frac{h_0}{h_1},a,\frac{b_0}{b_1});$
$(funct^{DH}_{ZK,out},\_) \leftarrow F^{DH}_{ZK}((h,a,b),((in_1,in_2,in_3),r));$
$\_ \leftarrow assert_{spmf}(funct^{DH}_{ZK,out});$
$let\ l = \frac{b_0}{h_0^r};$
$return((if\ l = 1\ then\ False\ else\ True),(h_0,h_1,a,b_0,b_1))\}$

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

Fig. 7. The simulators for Party 2

show the final result in the asymptotic setting. In particular in the asymptotic setting we can make the assumptions on the DDH advantage in full. We introduce the security parameter as described in Section IV-E and show security for Party 1 as follows.

Theorem 13: Assume that $\forall i \in \{0,1\}$ we have $negligible(\lambda n.DDH\_adv_{orig}(n,\mathcal{A}'(DDH\_\mathcal{A}_{\sigma=i}(\mathcal{A},D))))$ and $negligible(\lambda n.DDH\_adv_{orig}(n,DDH\_\mathcal{A}_{\sigma=i}(\mathcal{A},D)))$ then we have

$$negligible(\lambda n.mal_{adv_1}(n,(m_0,m_1),\sigma,z,S_1(n),\mathcal{A},D)).$$

Theorem 13 shows the advantage of a distinguisher distinguishing the real and ideal views of Party 1 is negligible.

3) Party 2: To show security for Party 2 we construct two simulators, S_{1,P_2}, S_{2,P_2} that interact with the ideal model such that the output distributions of the real and ideal model are equal. The simulators can be seen in Fig 7. In [13] (p191) the simulator is defined using only one part, we split it in to two as the definition of the ideal world ([13], p 23) requires.

To show equality between the real and ideal views we consider the cases on $l = \frac{b_0}{h_0^r}$ (constructed by S_{1,P_2}): $l = 1$, $l = g$, $l \notin \{1,g\}$. Such a case split is easy to reason about on paper, however due to l being defined inside a probabilistic program (the ideal view, mal_{ideal_2}) we cannot easily access it to perform the case split.

To make the case split easier we introduce a *case separation* technique. This allows us to more easily prove a property on a probabilistic program where the cases on a bound variable need to be considered. We describe our method informally here.

We isolate the case splitting variable by defining a new program that replicates the original program from the point the variable is introduced; the variable is now an input to a probabilistic program. Case splitting on inputs is natural so we prove the required property on the new program noting that some contextual assumptions may be needed in the statement. Using this statement we prove the required property on the original probabilistic program by recombining the newly defined program and the property proven about it and the original program.

In our proof we define $mal_{ideal\_end_2}$ that describes the end of the ideal view's probabilistic program, beginning at a point where l can be taken as an input. We also define the analogous end probabilistic program for the real view ($mal_{real\_end_2}$) and show the two programs are equal by case splitting on l.

Lemma 10: Assume that $\frac{b_0}{b_1} = (\frac{h_0}{h_1})^r$ and $h_0,h_1,b_0,b_1,m_0,m_1 \in G$ then we have

$$mal_{ideal\_end_2}((m_0,m_1),l,(h_0,h_1,g^r,b_0,b_1),\mathcal{A}_3) = mal_{real\_end_2}((m_0,m_1),(h_0,h_1,g^r,b_0,b_1),\mathcal{A}_3)$$

The proof of Lemma 10 is involved however the case split (on l) can be made easily which allows for a natural structure to the proof. The assumption on l is contextual information from the first part of the ideal view. Using Lemma 10 we show security for Party 2.

Theorem 14: Assume $m_0,m_1 \in G$ then we have

$$mal_{inf\_th_2}((m_0,m_1),\sigma,z,(S_{1,P_2},S_{2,P_2}),\mathcal{A}).$$

To prove Theorem 14 we show the real and ideal views are equal up to the point of l being introduced. This fact, together with Lemma 10, is used to show equality between the views. Thus we have shown security for Protocol 4.

We note Theorem 14 could be proven without Lemma 10 by manipulating the probabilistic programs, however this would have required more subtleties and low level reasoning. In fact we proved Lemma 2 without our *case separation* technique to provide a comparison of the methods. Our technique does not necessarily reduce the size of the proof however it does reduce the technicality of the proof.

VII. Conclusion

In this paper we have studied the semi-honest model: we considered OT^1_2 and formalised a construction using ETPs and HCPs, and instantiated it for the RSA collection. We considered OT^1_2 as a building block for OT^1_4 and showed how we use the modularity available in Isabelle to prove the GMW protocol secure. In doing so we formalised secret sharing schemes. Finally we provided the first fully formalised definitions for security in the malicious setting and proved secure an OT^1_2 protocol with respect to them.

This work has laid the foundations and made strong advances in the formalisation of both the semi-honest and malicious settings. In the semi-honest setting others can use our framework of definitions and multiple examples to conduct further formal proof. For example we have significantly reduced the workload required to prove instantiations of the OT^1_2 using HCPs and have shown how modular proofs can be considered. In the malicious setting we give definitions and a substantial security proof, that we believe will lower the bar for proving other protocols secure in the malicious model inside CryptHOL.

A major advantage of using Isabelle/HOL compared to other proof assistants for cryptography is the large amount of formalised mathematics available. For example, in proving the instantiated OT_2^1 using RSA we were able to prove a particular variant of Fermat's Little Theorem in a matter of lines (20 lines of proof) rather than prove the result from scratch.

Achieving a proof of security for malicious GMW would require a large proof effort. Significant extensions towards a formalisation of Zero Knowledge would need to be made to [7] as well as extending this work and [6] to the n party setting. The latter would likely be achieved by first formalising more basic n party protocols in the semi-honest model.

References

[1] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, and Vitor Pereira. A fast and verified software stack for secure function evaluation. In *ACM Conference on Computer and Communications Security*, pages 1989–2006. ACM, 2017.

[2] David Aspinall and David Butler. Multi-party computation. *Archive of Formal Proofs*, 2019, 2019.

[3] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *IACR Cryptology ePrint Archive*, 2017:753, 2017.

[4] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2008.

[5] 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. Secure multiparty computation goes live. In *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2009.

[6] David Butler, David Aspinall, and Adrià Gascón. How to simulate it in Isabelle: Towards formal proof for secure multi-party computation. In *ITP*, volume 10499 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2017.

[7] David Butler, David Aspinall, and Adrià Gascón. On the formalisation of Σ -protocols and commitment schemes. In *POST*, volume 11426 of *Lecture Notes in Computer Science*, pages 175–196. Springer, 2019.

[8] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[9] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[10] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[11] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[12] Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. Computer-aided proofs for multiparty computation with active security. In *CSF*, pages 119–131. IEEE Computer Society, 2018.

[13] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.

[14] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. *IACR Cryptology ePrint Archive*, 2016:46, 2016.

[15] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[16] Andreas Lochbihler. CryptHOL. *Archive of Formal Proofs*, 2017.

[17] Andreas Lochbihler. Probabilistic functions and cryptographic oracles in higher order logic. In *ESOP*, volume 9632 of *Lecture Notes in Computer Science*, pages 503–531. Springer, 2016.

[18] Andreas Lochbihler and S. Reza Sefidgar. Constructive cryptography in HOL. *Archive of Formal Proofs*, 2018.

[19] Andreas Lochbihler, S. Reza Sefidgar, and Bhargav Bhatt. Game-based cryptography in HOL. *Archive of Formal Proofs*, 2017.

[20] Ueli M. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[21] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[22] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.

[23] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.